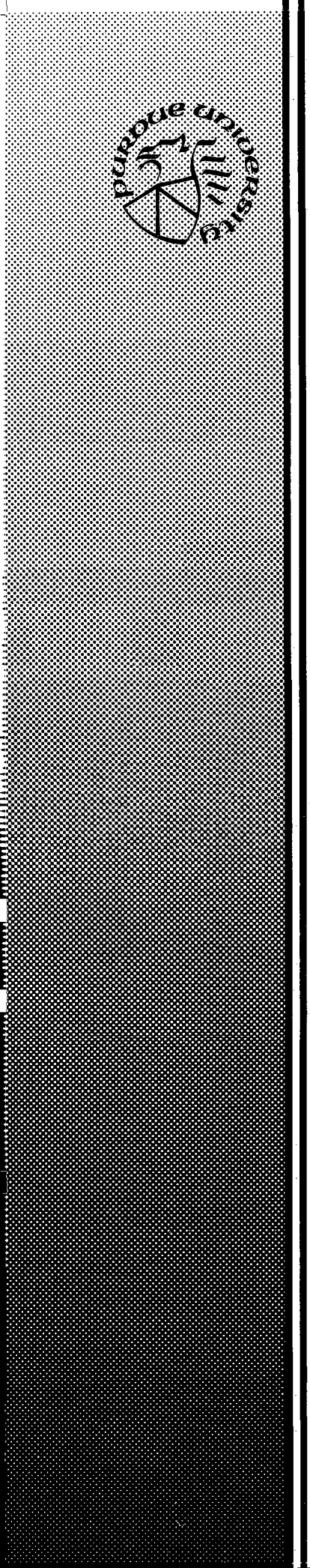
11-1-1989

The PSEIKI Report -- Version 3. Evidence Accumulation and Flow of Control in a Hierarchical Spatial Reasoning System

K. M. Andress
Purdue University

A. C. Kak
Purdue University



The PSEIKI Report -- Version 3

Evidence Accumulation and Flow of Control in a Hierarchical Spatial Reasoning System

K. M. Andress
A. C. Kak

TR-EE 89-35
November 1989

Robot Vision Lab

School of Electrical Engineering
Purdue University
West Lafayette, Indiana 47907

The PSEIKI Report -- Version 3

Evidence Accumulation and Flow of Control in a Hierarchical Spatial Reasoning System

K. M. Andress and A. C. Kak

TR-EE-89-35
November 1989

Robot Vision Lab
School of Electrical Engineering
Purdue University
West Lafayette, IN 47907

ACKNOWLEDGEMENT

Mobile robot navigation experiments with PSEIKI would not have been possible without active collaboration provided by a number of researchers in the Robot Vision Lab. Matt Carroll was in charge of of the hardware aspects of these experiments. Jeff Lewis took care of the software development required for the on-board processor and communications between the robot and the off-board processor. Carlos Lopez-Abadia developed the routines for camera calibration. Barry Roberts helped with the software and hardware for ultrasonic collision avoidance. Min Meng and Akio Kosaka performed basic experiments that helped us understand the poor quality of odometry on the robot.

This work was supported by the Army Research Office.

TABLE OF CONTENTS

Page

Page

ABSTRACT

A fundamental goal of computer vision research is the development of systems capable of carrying out scene interpretation using knowledge of the expected scene. Here, we describe PSEIKI, a framework for expectation-driven interpretation of image data. PSEIKI performs expectation-driven processing by matching elements, such as edges and regions, detected in an image with model-elements from a supplied expected scene. PSEIKI builds abstraction hierarchies in image data using cues taken from the supplied abstractions in the expected scene. Hypothesized abstractions in the image data are geometrically compared with the known abstractions in the expected scene; the metrics used for these comparisons translate into belief values.

The Dempster-Shafer formalism is used to accumulate beliefs for the synthesized abstractions in the image data. For accumulating belief values, a computationally efficient variation of Dempster's rule of combination is developed to enable the system to deal with the overwhelming amount of information present in most images. This variation of Dempster's rule allows the reasoning process to be embedded into the abstraction hierarchy by allowing for the propagation of belief values between elements at different levels of abstraction. PSEIKI has been implemented as a 2-panel, 5-level blackboard in OPS83. The operation and implementation of the blackboard's knowledge sources are described in detail. Control aspects of the blackboard's scheduler and distributed monitor are also described.

Finally, an experiment in which PSEIKI was used to aid in the navigation of an autonomous mobile robot will be described. PSEIKI was used to provide sensory feedback to update the estimates of the robot's position and orientation as it traveled in a known environment.

CHAPTER 1

INTRODUCTION

A fundamental goal of computer vision research is the development of systems capable of carrying out scene interpretation using domain knowledge. However, if the domain knowledge is programmed directly into the systems, they tend to become too domain specific and are capable of solving problems of narrow scope. Given the amount of effort it takes to program such systems, their payoffs tend to be rather limited. Some computer vision systems are able to remain domain independent by encoding domain knowledge as data. These systems do not need to be reprogrammed when applying them to a new application domain; one merely has to encode the new application's domain information into the appropriate format to allow these systems to function. It is in this vein that PSEIKI† was created. PSEIKI's domain knowledge is encoded in the form of symbolic description the scene expected to be visible in the image undergoing interpretation. PSEIKI was designed to be a domain-independent tool for expectation-driven scene interpretation; it is intended to be used by higher-level, domain-specific systems. PSEIKI can be used in a number of application domains.

PSEIKI originally was developed to aid navigation of an autonomous mobile robot as it traveled in a known environment between two specified points [KakRob87]. As the robot travels from its initial position to its goal position, errors in its hypothesized position and orientation accumulate to such a point that the possibility of a navigational error arises (e.g. the robot may run into a wall if its error in position is large enough). In this

† The acronym PSEIKI stands for a Production System Environment for Integrating Knowledge with Images. The evolution of the system can be followed by reading [AndKak87a], [AndKak87b], [AndKak88a] and [AndKak88b].

application, PSEIKI provides sensory feedback to update the estimates of the robot's position and orientation as it travels along its path. PSEIKI integrates vision information observed by a robot-mounted camera with the scene expected to be visible from the camera given the robot's hypothesized position and orientation. Once PSEIKI is used to merge data from the expected scene and the image, triangulation then is used to update the position of the robot in the world coordinate frame. This application of PSEIKI will be explored in greater detail in chapter 10.

PSEIKI can be used for expectation-driven interpretation of vision data in other domains in which a good estimate of the expected scene is available. For example, in the navigation of a self-guided munition, PSEIKI could be used to compare an image of the terrain with a map of the terrain; the results produced by PSEIKI then could be used to yield an updated fix on the location of the munition. In a target recognition system, PSEIKI could be used to verify the output produced by a error-prone low-level pattern recognition system. The hypothesized identity produced by the recognition system could be used to generate the expected scene for PSEIKI; if PSEIKI determined that the observed image did not match the expected scene to a significantly high degree, then the hypothesized identity would be deemed incorrect. In industrial applications, PSEIKI could be used to monitor the progress of assembly robots. At key times in the assembly sequence, PSEIKI could be used to verify that the process is proceeding normally by comparing an image of the assembly cell with a description of the scene expected to be visible in the cell. In this application, the CAD information describing the part being assembled could be used to generate the expected scene. Such verification systems are expected to play an important role for robotic assembly cells in the future.

PSEIKI performs expectation-driven processing by matching elements, such as edges and regions, detected in an image with model-elements in a supplied expected scene. The match information generated by PSEIKI is expressed by labeling the image-elements with the identities of the corresponding model-elements; a belief value indicating the confidence of the match is attached to each label. Fig. 1.1 shows an example of PSEIKI's matching of image and expected scene information for a mobile robot traversing a known network of sidewalks. Panel (a) of this figure shows an expected scene for a camera mounted on the mobile robot. If PSEIKI's preprocessor produces the edges shown in panel (b) from the scene's vision data, then PSEIKI would produce an output similar to the one in panel (c). This panel shows the labels attached to the edges in the scene interpretation with highest belief; the belief values associated with the overall

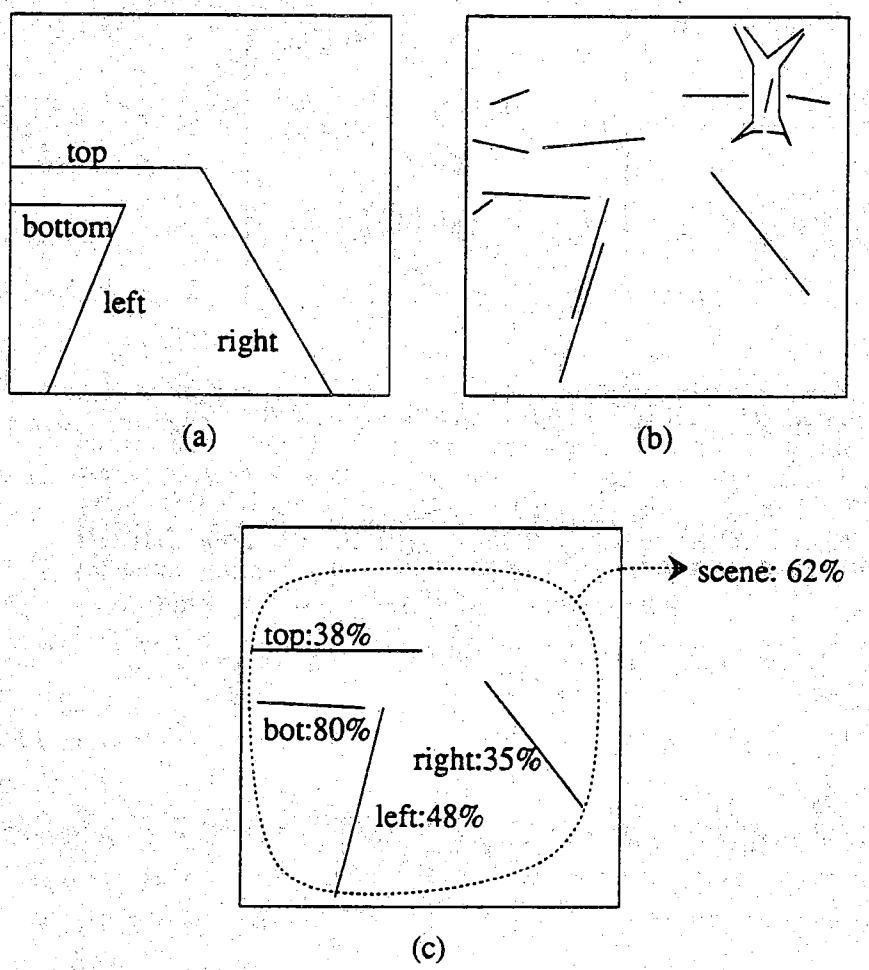


Figure 1.1 This figure shows typical images used by PSEIKI. The image in panel (a) shows an example of an expected scene with edges labeled. Panel (b) shows a simple example of the output of an edge based preprocessor which PSEIKI would use as input data. Panel (c) shows the final output of PSEIKI, with labeled edges and the labels' belief values in the most plausible interpretation of the scene. The confidence value attached to the overall interpretation of the scene is also shown.

scene interpretation and with each of the edge matches are also shown. For example, the label 'right:35%' means that PSEIKI has found the expected-scene edge labeled 'right' in panel (a) to be compatible with the lower right edge in panel (b) with a belief of 35%. In this case, the rest of the belief, 65%, would be apportioned either to this particular label being incorrect or to the system professing ignorance about this edge's label. The reader might note that the edge labeled 'top:38%' actually corresponds to two edge segments in panel (b). This merger of nearly compatible image edges is one consequence of various tests PSEIKI makes for internal geometric consistencies in the vision data.

Although the above example demonstrates PSEIKI's processing using edges, PSEIKI is able to reason about data at higher levels of abstraction. For example, regions in the image, represented by the edges forming their borders, are matched with regions in the expected scene. Since the image preprocessor can deposit only low-level information, PSEIKI forms the higher-level constructs by grouping the low-level elements using cues taken from the supplied abstractions in the expected scene. For example, if PSEIKI's low-level preprocessor provides edge information to the system, then PSEIKI would form a face by grouping edges together if they had compatible labels and met appropriate geometric constraints. The following list enumerates the levels of data abstraction present in PSEIKI and describes the data residing on each level.

Level 1: Vertices -- Vertices are used to define the endpoints of the edges from level 2 of the hierarchy. They can be expressed either in world or image coordinates depending on the type of data they represent.

Level 2: Edges -- The elements on this level represent straight line segments. They can be used to represent edges detected by the image preprocessor or can be used to form the boundaries of the faces stored in the next level of the hierarchy. Arbitrary curves are represented approximately by sequences of edge elements; this approximate representation of curves restricts PSEIKI's domain to polyhedral data.

Level 3: Faces -- The elements on this level represent 2 dimensional constructs. In image data, a face corresponds to a region in the image; in model data, each polygonal face represents a visible surface of an object in the expected scene.

Level 4: Objects -- Each element on this level corresponds to a distinct physical object defined by its boundary faces from level 3.

Level 5: Scenes -- The entire scene (expected or observed) is represented on this level. The scene is defined as the collection of all objects in level 4 of the hierarchy. At the end of processing, the scene-level image element with highest belief is chosen as the final scene interpretation. The belief in the label of this element is interpreted as the confidence in the entire matching process and is used to determine if the matching process has succeeded.

Fig. 1.2 shows how a simple scene, a single block, can be broken down hierarchically. Each element in this hierarchy is defined by its parts on lower levels. This figure demonstrates how an object can be defined in terms of its bounding faces and how a face can be defined by the group of edges which form its border.

The Dempster-Shafer theory is used to accumulate evidence on the certainty of the matches between image elements and expected scene elements. This formalism has the advantage of allowing the explicit expression of ignorance about an element's label if that element does not match any model element to a sufficiently high degree.† To overcome the exponential explosion usually associated with the Dempster-Shafer formalism, a computationally efficient variation of Dempster's rule is used to combine evidence about the labels. This variation of Dempster's rule also allows the reasoning process to exploit the hierarchical nature of the integration task. For example, the belief value associated with the top level of the hierarchy is considered to be the confidence in the entire matching process; if this belief value does not exceed a threshold, the matches found are rejected.

PSEIKI exploits geometric relationships between data-elements at the above levels of abstraction in the reasoning process. Initial matches between image data and model data are formed by noting geometric relationships between image-elements and model-elements. For example, an image-edge will be matched with the model-edge that comes the closest (in some sense) to lying along the same line in the world coordinate frame. To find the match partner of an image-edge, PSEIKI measures the degree of collinearity between that edge and all the model-edges in its vicinity; it then chooses as the match partner the model-edge with which the image-edge is most collinear. The belief of the

† A Bayesian would probably insist that one could distribute belief evenly amongst all possible labels when an image element can not be matched with any model element. We do not dispute that. However, when belief values must be generated from ad-hoc measures, expressing ignorance by withholding belief becomes a convenient aspect of evidence accumulation -- something that is not allowed in a Bayesian formalism.

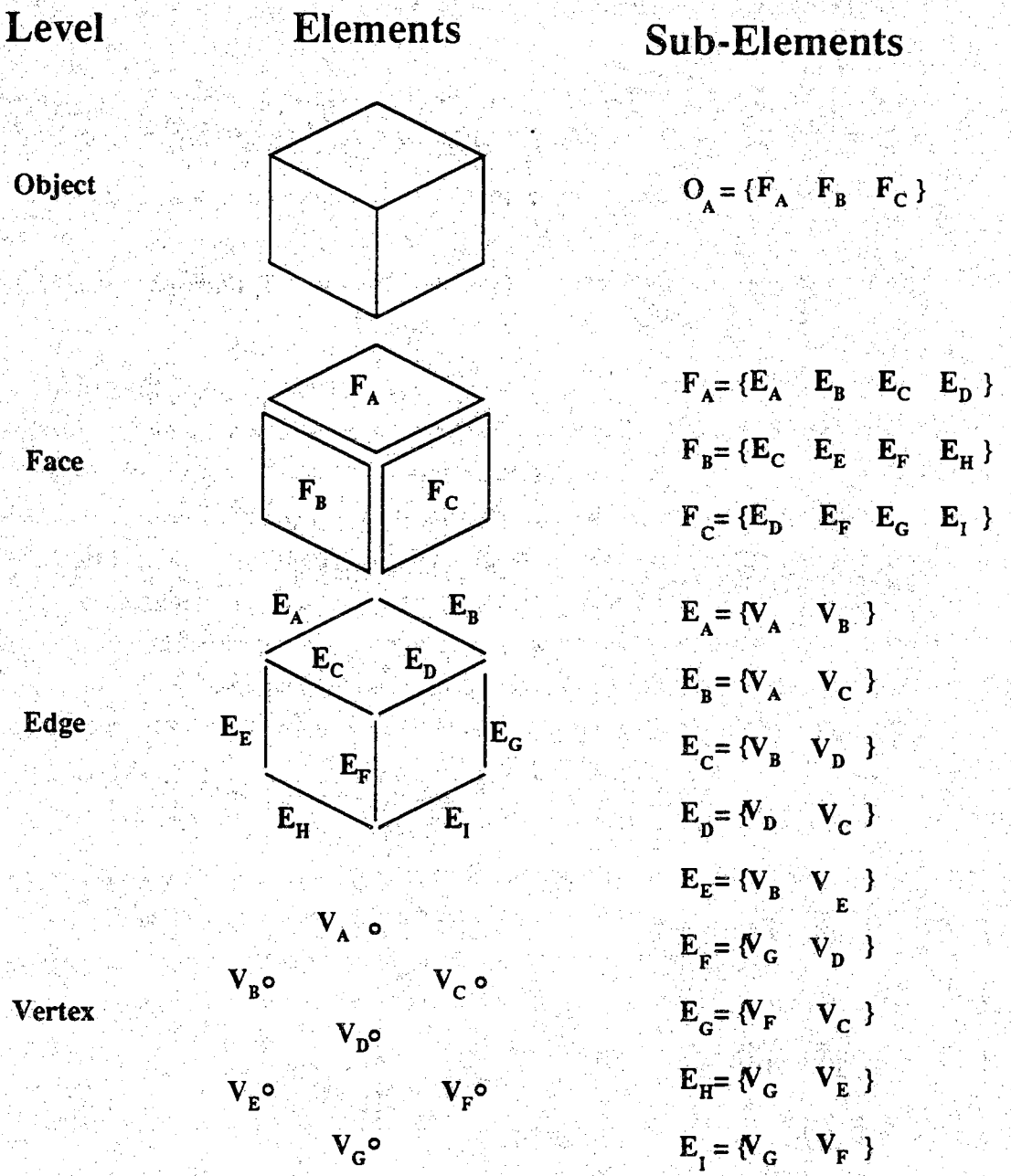


Figure 1.2 This figure shows how a simple scene can be broken down hierarchically into objects, faces, edges and vertices.

match then is made proportional to the degree of collinearity between the two edges.

After the initial matches are made, the beliefs associated with the matches are updated based on the extent to which image elements satisfy spatial constraints, dictated by the model information. In general, two metrics are required to measure the degree to which image-elements meet these constraints. The two metrics must provide measures of compatibility and incompatibility between image-elements given the spatial relationships amongst their matched model elements. The compatibility and incompatibility metrics provide evidence that an element's label is correct or incorrect, respectively, based on the degree to which the model generated constraints are satisfied. For example, two edges that have the been matched with the same model-edge should lie approximately along the same line. Thus, for edge-elements with the same label, the compatibility metric measures the degree to which the two edges lie along the same line. This collinearity metric is closely related to the measure used to establish initial edge labels. The edge-level incompatibility metric measures the degree to which two edges do not lie along the same line. Of course, different (in)compatibility metrics must be used at each level of abstraction. For example, the metrics that are used to compute the (in)compatibility between two faces on the data panel measure the degree to which their relative positions match the relative positions of their model faces (based on the centroids).

An important aspect of evidential reasoning in PSEIKI is the propagation of beliefs up and down the abstraction hierarchy. The propagation of belief values towards the higher abstraction levels is based on the rationale that any evidence confirming a data element's label should also provide evidence that its parent's label is correct. Propagation of beliefs to lower levels is based on the intuitive idea that if an element is mislabeled then its constituent elements most likely are mislabeled (for example, a face and its constituent edges).

PSEIKI has been implemented in OPS83 as a 2-panel / 5-level blackboard, as shown in Fig. 1.3. The left panel, called the model panel, holds the abstraction hierarchy for the expected scene; the model data is deposited onto all levels of this panel by the expected scene generator. The right panel, called the data panel, holds the abstraction hierarchy for the image data. Data is deposited onto the lowest levels of this panel by the preprocessor; data elements on the upper levels of this panel are created in the course of blackboard processing. Each level in the blackboard corresponds to one of the levels of data abstraction discussed earlier. Thus, each blackboard panel contains the following

abstraction levels: scenes, objects, faces, edges and vertices. Each element on the blackboard, except for vertices, is defined by a finite collection of lower-level elements.

PSEIKI has four knowledge sources (KSs) that it uses to establish correspondences between image-elements and model-elements: labeler KS, grouper KS, splitter KS, and merger KS. The grouper KS determines which data-elements at a given level of the hierarchy should be grouped to form a data-element at a higher level. For example, if a set of edges is believed to form the border of a geometrically significant region in the image, then the grouper KS would group them together into a face. The merger KS also groups elements; however, its job is to merge multiple elements at a given level and retain the grouped information at the same level. For example, the merger KS may group together a series of short edge segments into a longer segment, or a set of faces into a single larger face, if it is believed that the low-level preprocessor incorrectly fractured those data elements. The splitter KS performs the opposite action of the merger KS; it splits a single element on the blackboard into multiple smaller elements†. Its main task is to guarantee that the grouper KS does not include incompatible data elements in a single grouped element. The labeler KS has the responsibility of establishing model to data correspondences at all levels of the blackboard and of accumulating evidence on the validity of those correspondences. Each of these KSs can operate at any level of the blackboard by using level-specific actions.

As was mentioned before, the input image is first preprocessed and then deposited onto the lowest levels of the data panel. The type of preprocessing performed by a low-level systems determines the blackboard levels on which the data is deposited. The symbolic information produced by edge based preprocessors is deposited directly at the vertex and edge levels of the data panel. On the other hand, for preprocessors that produce region type outputs, the additional information is fed directly onto the face level of the data panel††. This additional input has been depicted by a dashed line in Fig. 1.3. Even

---

† For those familiar with our earlier publications on PSEIKI, the merger and the splitter KSs in the current implementation are a 'generalization' of the data-reduction KS in the earlier version of the system. The data-reduction KS could operate only at the edge level of the blackboard and its function was to merge edge segments into longer edges and to delete short segments. On the other hand, the merger KS and the splitter KS can merge and split information at all levels of the blackboard. They also are scheduled in a more integrated fashion during blackboard processing.

†† Although the original version of PSEIKI, as reported in [AndKak88a], could accept only edge level information from the preprocessor and the expected scene generator, the newer version reported here requires that expected scene information be deposited on all levels of the blackboard; it can also handle region-based image inputs directly.

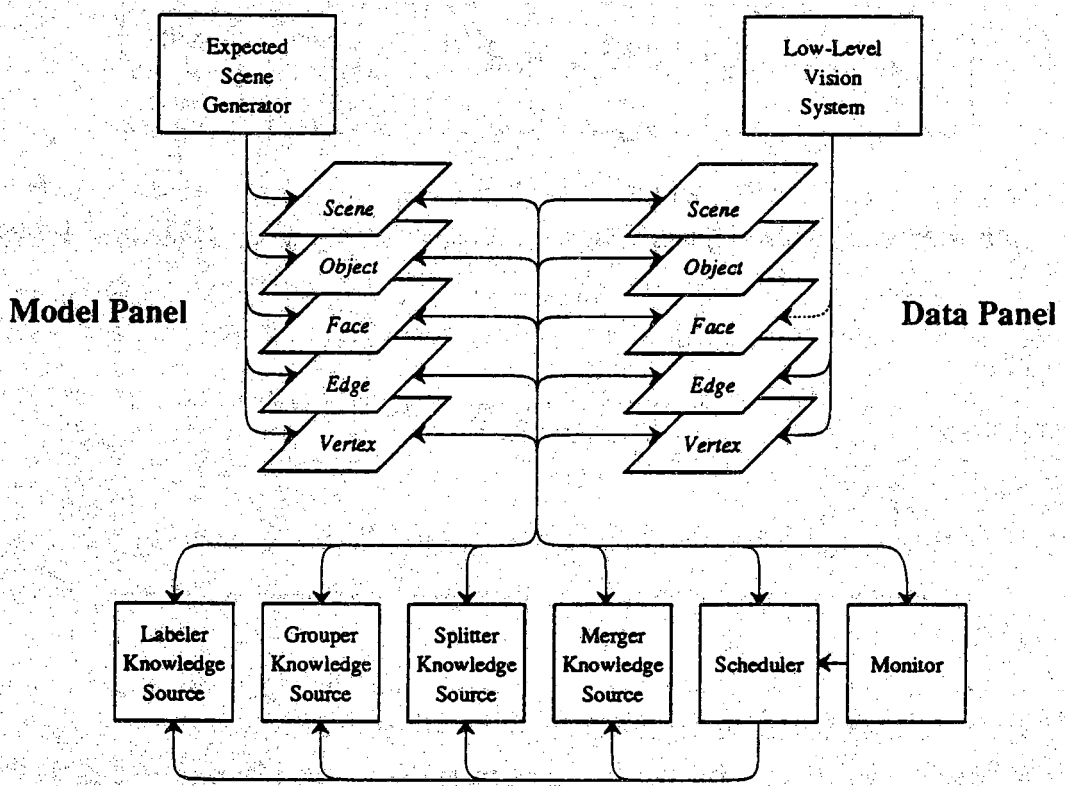


Figure 1.3 This figure shows the current configuration of PSEIKI's architecture.

when image data is presented to PSEIKI in a region based form, the system still exploits edge level information by treating the boundaries between regions as edges and matching them with edges in the expected scene. The edge level information is not ignored because much of the information about a region is contained in the shape of its borders; this border information is stored on the edge level. Model data is deposited onto all levels of the blackboard because it is assumed that perfect knowledge of the expected scene is available.

Work related to PSEIKI will be discussed in the next chapter; a survey of some previous knowledge-based computer vision systems will be presented there. Chapter 3 discusses the type of preprocessing that must be carried out before an image can be presented to PSEIKI; in this chapter, the data structures used for describing the image elements also will be shown (the same data structures are used for model elements). Chapter 4 will focus on the generation of expected scene information and will discuss several CAD systems we have used for this purpose. Chapters 5, 6, 7 and 8 are used to describe PSEIKI in detail. Chapters 5 and 6 present the techniques used in the labeler KS to generate and accumulate evidence for correspondences between the data and the model elements. In particular, chapter 5 describes a hierarchical evidence accumulation scheme based on the Dempster-Shafer framework. Chapter 6 demonstrates how geometric constraints can be used to generate evidence about the matches found between elements. Chapter 7 gives a detailed description of blackboard operation; the operation of the individual KSs, the scheduler and the monitor will be discussed here. Chapter 8 discusses the implementation of the blackboard in OPS83; the data structures and some representative rules will be shown. Complexity issues of blackboard processing are addressed in chapter 9. An example application of PSEIKI is presented in chapter 10. This chapter describes an experiment where PSEIKI is used update the hypothesized position of a mobile robot as it traverses a known network of hallways. Finally, some possible improvements to PSEIKI are presented in chapter 11.

CHAPTER 2

RELATED WORK ON SPATIAL REASONING

In this chapter, we will briefly survey what has been done to date in the development of knowledge based systems for image understanding. We will describe the salient characteristics of each system, including their overall task, flow of control, use of any inexact reasoning schemes and any methods used to provide domain independence.

An early model-based image understanding system, ACRONYM, is described by Brooks in [Bro81]; the task of this system consists of finding instances of known objects in an image. To perform object identification, the system first builds an *Prediction Graph* that specifies information about objects that could be in the image; generalized cones are used to represent these model objects. The nodes in the graph represent predictions of image features; the arcs specify relations between features. The system then builds a *Picture Graph* of the image and identifies instances of objects in the image by matching nodes of the Prediction Graph with sets of nodes in the Picture Graph. The objects in the Prediction Graph are represented in *slot - filler* structures where any slot that can accept numeric values can also accept algebraic constraints expressed as inequalities. The system then can manipulate these constraints and determine if they are met by properties of objects detected in the image. ACRONYM uses only backward chaining in the matching process and does not incorporate inexact reasoning. Because ACRONYM's model information is stored completely in the Prediction Graph, its application domain can be changed by replacing the information in the graph with model information from the new domain.

The SIGMA image understanding system for aerial interpretation was first described in [MatHwa85] and later developed in [HwaDav85]. The system is composed

of three main parts, a Low-Level Vision Expert for knowledge-based image processing, a Model Selection Expert for selection of appearance models, a Geometric Reasoning Expert that performs the systems spatial reasoning. The system represents its object classes hierarchically using frames, each slot of a frame contains a production rule about the object class. The system's flow of control integrates bottom-up and top-down reasoning into an integrated reasoning process. The system is able to integrate hypotheses about specific objects in the scene by clustering related hypotheses and verifying the "composite hypothesis." The system does not use uncertain reasoning, but instead is able to control its focus of attention based on the strength of a situation.

Another aerial interpretation system is described by Nagao and Matsuyama [NagMat80]; the system is based on the blackboard architecture and uses multispectral images in the interpretation process. To accomplish the interpretation task, the system first performs a global survey of the entire image and labels regions without using domain specific knowledge. The *characteristic regions* that it finds, such as water, vegetation, roads, etc., then are used to generate context information for further blackboard processing. This processing consists of a detailed analysis of local areas in the scene using context information provided by the characteristic regions and applying context specific object detection subsystems.

SPAM, a system designed by McKeown, Harvey and McDermott also is an aerial image interpretation system [MckHar85]. The system originally was constructed to interpret airport scenes but has been expanded with a rule generator so that it now can interpret scenes from other domains. SPAM uses confidence values to aid in labeling and can manipulate these values based on the consistency of the various labels.

VISIONS (Hanson and Riseman) is a blackboard expert system designed to analyze color images [HanRis78]. The system uses a flexible control scheme, hierarchical scene representation, and a number of knowledge sources to accomplish the scene interpretation task. VISIONS is domain independent, but schemas can be used to tune the system for a particular application.

Nazif and Levine describe an expert system based image segmenter in [NazLev84]; the system was designed to provide a framework that would allow the combination of edge, region and area based segmentation techniques. With these segmentation techniques, the segmenter can split and merge regions, link and break edges and operate on image areas based on features of the elements. The system is rule-based and stores its

rules in a global long term memory; the image data undergoing segmentation is processed in a short term memory. The expert system, which contains a set of metarules, can focus its attention on interesting areas of the image.

PSEIKI differs from the above knowledge-based systems in the following three main areas: First, PSEIKI's task differs from those of previous systems. Most of the other systems were designed to find object instances in the image and, through such discoveries, to arrive at a global interpretation of the image. PSEIKI's task is limited to matching image data with expected scene information and indicating the belief in the matches. Thus, a higher level system is need to make a global interpretation of the scene content based on the match information produced by PSEIKI.

PSEIKI also can be contrasted with SPAM and SIGMA, and to a certain extent VISIONS, in that its domain information is not embedded in the inference engine. For example, SPAM uses rules containing airport design knowledge when interpreting airport scenes. On the other hand, PSEIKI's domain information is encoded entirely in the form of the graphic rendition of the expected scene. Context-cues also have been used extensively in past computer vision systems. For example, if SIGMA detects a driveway in an image, it then would search for a house and for roads connected to the driveway. Because PSEIKI is provided with a good estimate of the expected scene, it does not have to perform inferences of this type. Although context-cues are indispensable for scene interpretation because they make deductions more powerful, adding rules to the inference engine to exploit the context-cue necessarily introduces some domain dependence. Therefore, it is our philosophy to separate the expected-scene/image matching from the formation of an overall interpretation of the scene. If the use of context-cues is desired by a system using PSEIKI, then it is up to the higher level system to provide PSEIKI with expected scene data incorporating the information contained in the cues.

PSEIKI also differs from previous systems in its method of performing inexact reasoning. Many systems, including ACRONYM, SIGMA and the system by Nazif and Levine use no uncertain reasoning in the image interpretation process. Because of the overwhelming amount of data in an image, most of the inexact reasoning schemes used in the past have employed simple combination schemes in order to keep from becoming bogged down in certainty value computations. On the other hand, inexact reasoning in PSEIKI is based on the Dempster-Shafer formalism in a tangled hierarchical space. The use of a hierarchy curtails the number of uncertainty calculations and is made possible by the use of the blackboard architecture.

CHAPTER 3

PREPROCESSING OF INPUT VISION DATA

The image to be interpreted must first be converted into a symbolic form before it can be deposited on the lowest two or three levels of the data panel of the blackboard. This chapter will focus on the preprocessing steps used to convert image data into the required symbolic form; the format of the input data also will be described. Both the image preprocessor and the expected scene generator are required to present their data to PSEIKI in this format. The same format also is used to output the match information produced by PSEIKI.

The chapter will describe two image preprocessors. The first preprocessor uses an edge-tracking scheme to generate an edge-based symbolic description of the input image. The input data presented by this preprocessor is deposited on the edge and vertex levels of the blackboard. The second preprocessor employs a region-growing scheme to produce a region-based symbolic description of the input image. This preprocessor also feeds information at the face level in addition to that at edge and vertex levels. The data on the face level represents the regions extracted by the preprocessor; the borders between these regions are represented as edges. Finally, the endpoints of the edges are input at the vertex level.

The two preprocessors described here are presented only as examples of systems that can generate input data. Because they both use well known techniques, they will not be described in great detail. Furthermore, no claim of optimality is made for any of the systems presented. In fact, for PSEIKI to be a truly general expectation-driven vision system, it should be robust enough to overcome any peculiarities of these or most other low-level preprocessors. Thus, if improved low-level preprocessing techniques become

available in the future, PSEIKI should be general enough to use the output produced by the new preprocessors†.

3.1. Format of Input Data

PSEIKI expects to see its input data as an ASCII text file with each line corresponding to a separate data element, as shown in Fig. 3.1. The fields used in the data files are self-explanatory. The first field following the '+' on a line specifies the level of the blackboard onto which the element is deposited. All other fields are specified by keyword - data pairs; the data part of some fields can hold multiple values. For example, the data part of the children field can specify that an element has more than one child. The *id* field is used to specify a unique identification number for a data element; each element on the blackboard is referenced via its ID number. The element's *children* field specifies the sub-elements that are used to build it; for example, an edge has two children -- its end vertices. If an element is a vertex, its location is specified via a field with three data elements, the *coordinate* field. If the vertex is located in three-space, then the data part of this field holds three values -- the x, y and z values of its location, respectively. However, if the location of the vertex is specified in the image plane, the first two data elements specify its column and row respectively; the third element is ignored. Any text appearing after a semicolon is considered to be a comment and is ignored. Besides the fields shown in Fig. 3.1, there are a number of optional fields that the low-level systems can use to provide additional information to PSEIKI. The *value* field can be used to provide PSEIKI with a level specific value; for example, this field can be used to indicate an edge's average strength or a region's average grey level. Likewise, the *size* field can provide PSEIKI with level specific size information (e.g. region area, edge length, degree of a vertex).

It is possible for PSEIKI's preprocessors to indicate that certain elements should not be used during blackboard processing. For example, the edges that form the border of the expected-scene or observed image do not have any physical significance in the real world and should not be subject to reasoning. These elements are flagged with negative ID numbers. When PSEIKI detects an element with a negative ID number, it removes

† [Bla89] describes a graphics tool that has been developed for debugging PSEIKI and testing its robustness.

```
+ object   id 1   children  2  3  4          ; object A

+ face     id 2   children  5  6  7  8       ; face A
+ face     id 3   children  7  9 10 12       ; face B
+ face     id 4   children  8 10 11 13       ; face C

+ edge     id 5   children 14 15             ; edge A
+ edge     id 6   children 14 16             ; edge B
+ edge     id 7   children 15 17             ; edge C
+ edge     id 8   children 16 17             ; edge D
+ edge     id 9   children 15 18             ; edge E
+ edge     id 10  children 17 20             ; edge F
+ edge     id 11  children 16 19             ; edge G
+ edge     id 12  children 18 20             ; edge H
+ edge     id 13  children 19 20             ; edge I

+ vertex   id 14  coordinates 1.0 1.0 1.0   ; vertex A
+ vertex   id 15  coordinates 1.0 0.0 1.0   ; vertex B
+ vertex   id 16  coordinates 0.0 1.0 1.0   ; vertex C
+ vertex   id 17  coordinates 0.0 0.0 1.0   ; vertex D
+ vertex   id 18  coordinates 0.0 1.0 0.0   ; vertex E
+ vertex   id 19  coordinates 1.0 0.0 0.0   ; vertex F
+ vertex   id 20  coordinates 0.0 0.0 0.0   ; vertex G
```

Figure 3.1 This is a sample data file demonstrating PSEIKI's input data file format.

the element from focus and uses the absolute value of the id field as the element's ID number. It is important that the preprocessors provide these border edges to PSEIKI because they are used to determine their parent faces' geometry (e.g. their convex hulls, centroids, etc.).

This format also is used to output the match results produced by PSEIKI. As a final step in PSEIKI's processing, the scene element on the data panel with highest belief is chosen as the final scene interpretation. This scene element and all of its descendents are output using the above format; however, two additional fields are used to store the match information. The *label* field is used to indicate the ID number of the model element with which each data element is matched. The other additional field, the *belief* field, is used to store a number between zero and one indicating the belief in the element's label.

3.2. An Edge Based Image Preprocessor For PSEIKI

Edge detection is a common technique used in image segmentation and other low-level image processing [RosKak82], [BalBro82]. However, the most common edge detection process, which consists of thresholding the output of a gradient type window operator, is incapable of generating input data directly for PSEIKI. This is due to the difficulty encountered when converting thick edges produced by this process to the symbolic form required by PSEIKI. Although iterative methods are available to reduce the widths of these edges, they are prohibitively time-consuming [RosKak82], [Ebe76], [BalBro82]. Ridge-tracking is another method that can be used for edge detection [WatArv87]. A variation of the ridge-tracking algorithm described in [Kim88] has been adapted to convert edges into a form usable by PSEIKI. A modification of the original algorithm was necessary due to PSEIKI's requirement that all of its input data be represented symbolically. The original algorithm's inability to find edge intersections also has been corrected in PSEIKI's preprocessor. There are a number of steps to the modified segmentation process.

1) First, a window-based gradient operator is applied to the image; the Sobel operator is used in the current system [RosKak82]. Since the ridge-tracking algorithm uses only gradient magnitude information, the direction of the gradient is not computed.

2) After the gradient operator is applied to the image, every pixel above a user-specified threshold is stored in a list; this list of pixels is called the *threshold list*.

Since the system only works on pixels in this list (usually between 5% and 10% of the total number of pixels), the required amount of work is drastically reduced.

3) To reduce the algorithm's noise sensitivity, all pixels in the threshold list are averaged with their eight closest neighbors.

4) The next step in the process consists of finding all edge endpoints; eventually, these pixels correspond to vertices on PSEIKI's blackboard. To find these elements, the notion of the *degree of one dimensional maximum* (DODM) is used. Each pixel has four pairs of neighbors -- horizontal neighbors, vertical neighbors, and neighbors in two diagonal directions. The DODM of a pixel is the number of pairs of neighbors in which both neighbors have lower values than the pixel itself. Fig. 3.2 demonstrates this concept; the DODM for the center pixel, "C", is defined to be the number of cases in which its value is greater than those of both of its two neighbor pixels, "N". The center pixel of the image neighborhood shown in Fig. 3.3 has DODM 2 since it has greater value than its four neighbors in the horizontal and vertical directions. All pixels in the threshold-list with DODM of three or four are considered to be edge endpoints.

5) It is in the next step in segmentation that the ridge-tracking process actually occurs. Two image structures are used to aid in this ridge-tracking process; these image structures are called the *edge* and *mark* arrays. The edge array is used to record the pixels that have been determined to be endpoints or parts of an edge. If the value of a pixel is nonzero in the mark array,

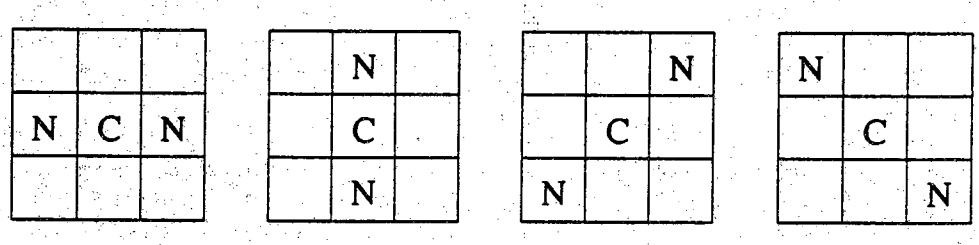


Figure 3.2 This figure demonstrates the concept of the Degree of one Dimensional Maxima (DODM). The DODM for the center pixels is defined to be the number of cases (1-4) in which the center pixel "C" is larger than both adjacent pixels "N" along a line.

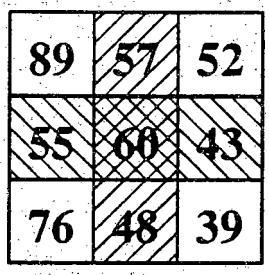


Figure 3.3 The DODM of this example image neighborhood is 2 because the center pixel has a larger value than its horizontal and vertical neighbors.

then the pixel is said to be marked and the tracker will not follow the edge onto that pixel. This technique is used to keep the tracker from backtracking onto pixels recently determined to be part of the edge. Another concept that is used in the tracking process is called the *current(i) pixel*; this is the ridge pixel that was determined, at time i, to be part of the edge. The tracking process is described below; Fig. 3.4 is used to demonstrate the operation of the tracking algorithm; it shows a subsection of an image containing a vertex pixel found in step (4).

5a) Let i = 0. Obtain an endpoint vertex found in step 4 of the process and designate this as the current(0) pixel. In the edge array, label this pixel as an endpoint and mark this pixel in the mark array (by setting the value of the pixel in the mark array to nonzero). In panel (a) of Fig. 3.4, the pixel with value 25 was designated as the current(0) endpoint pixel. It is shown in boldface to indicate that it has been designated in the edge array as an endpoint; it is shaded to indicate that it has been marked in the mark array.

5b) In the edge array, label the current(i) pixel (if $i \neq 0$) as an edge pixel and let $i = i + 1$.

5c) Choose the current(i) pixel in the following manner: If there is an unmarked endpoint or edge pixel adjacent to the current($i - 1$) pixel in the edge array, choose this unmarked pixel as the current(i) pixel, designate it as an endpoint in the edge array and stop the tracking process. Otherwise, find the next pixel in the edge by finding the

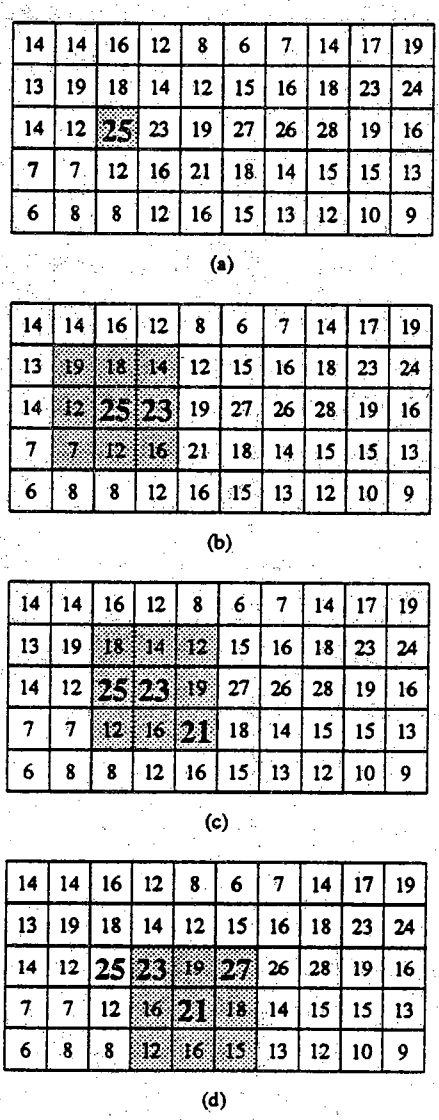


Figure 3.4 This figure demonstrates the marking of pixels in the ridge-tracking algorithm. The boldface pixels represent edge pixels and the shaded pixels are marked.

strongest unmarked pixel which is adjacent to the current(i – 1) pixel and has DODM ≥ 2. Label this pixel as current(i), add it to a list of pixels that denote the edge, and designate it as an edge in the edge array. If no pixel fits this description, then the edge "died;" designate the current(i – 1) pixel as an endpoint and stop the tracking process.

5d) If i ≥ 2 then unmark the current(i – 2) pixel and its eight neighbors.

5e) Mark the current(i – 1) pixel and its eight neighbors. Panels (b) - (d) of Fig. 3.4 show the status of the tracking process at this point in the procedure for i = 1, 2, 3, respectively. In each case, the block of marked pixels surround the pixel added to the edge on the previous cycle. Thus, the block in panel (b) surrounds the current(0) pixel, etc. Notice that current(i) pixel is always on the edge of the marked block, allowing the edge to be extended but preventing any backtracking.

5f) Go to step (5b).

The original algorithm never unmarked pixels after they were marked; this prevented the system from finding junctions between edges. By unmarking pixels when there is no possibility of the ridge-tracker backtracking onto freshly labeled edge pixels, these vertex pixels can be found. If the number of pixels in an edge is less than a user specified threshold, then the list is deleted and all pixels in the edge matrix are reset to their original state.

A few iterations of the tracker at step (5e) are shown in Fig. 3.4 to demonstrate how the tracking algorithm works. In this illustration, the pixels in boldface have been labeled as belonging to the edge. The shading denotes pixels that have been marked on the current iteration of the tracking algorithm.

6) The final step of the segmentation process is the fitting of piecewise-linear segments to the lists of edge pixels. This step is based on a process described in [DudHar73] and also used in [NavBab80]. This step also requires a user-specified parameter -- the maximum fitting error, E_{max}. In this process, a line, called the *model* line, is drawn between the two endpoints of an edge; then the edge pixels are followed (by traversing the list of edge pixels) and the distance between the individual pixels in the

edge and the model line is computed. If the distance between every pixel and the line is less then E_{max}, then the edge can be represented by the model line. However, if any pixels are greater than E_{max} away from the model line, then the pixel that is the farthest from the model line is considered to be a new endpoint and the line fitting algorithm is called recursively (once for each edge between the new endpoint and the old endpoints). The line fitting process is shown in Fig 3.5; in this example, the line-fitting process breaks the line into two piecewise linear segments.

The segmentation process, including the intermediate steps, is shown in Figs 3.6 and 3.7. Fig. 3.6 demonstrates the process when applied to an image typical of those taken by a mobile robot with downward pointing cameras. Fig. 3.7 demonstrates the process when applied to an industrial scene. In each of these two figures, panel (a) shows the original image; panel (b) shows the magnitude of the gradient as found by the Sobel operator, and panel (c) shows the edges that were traced by the ridge-tracking algorithm. Panel (d) shows the final output of the segmenter after it has converted the edges in panel (c) into piecewise-linear segments.

This preprocessor is fairly efficient due to the use of linked lists for representing edges. The segmenter was applied to a set of 512×480 test images; the system was able to segment an image (perform the Sobel operation, threshold, smooth, ridge-track and convert to symbolic form) in an average of 45 seconds on a SUN/3.

3.3. A Region Based Image Preprocessor For PSEIKI

As was mentioned before, PSEIKI can accept either edge-based or region-based symbolic descriptions of the input image. The preprocessor that currently is used to produce a region-based description of the input image is based on region growing ideas first advanced in [BriFen70] and developed further in [HorPav76]. The implementation described here uses the *quadtree* data structure that has become popular since the original algorithm was published in [HorPav76]. The quadtree data structure, a well known tool for representing binary images [Sam84a, Sam84b], has been extended in this application to represent greyscale images. There are a number of steps that the region growing process uses to generate the final segmented image. Figs. 3.10 and 3.11 demonstrate the region-growing process when it is applied to the sample images described in the previous section. The original images are shown in panel (a) of the two figures.

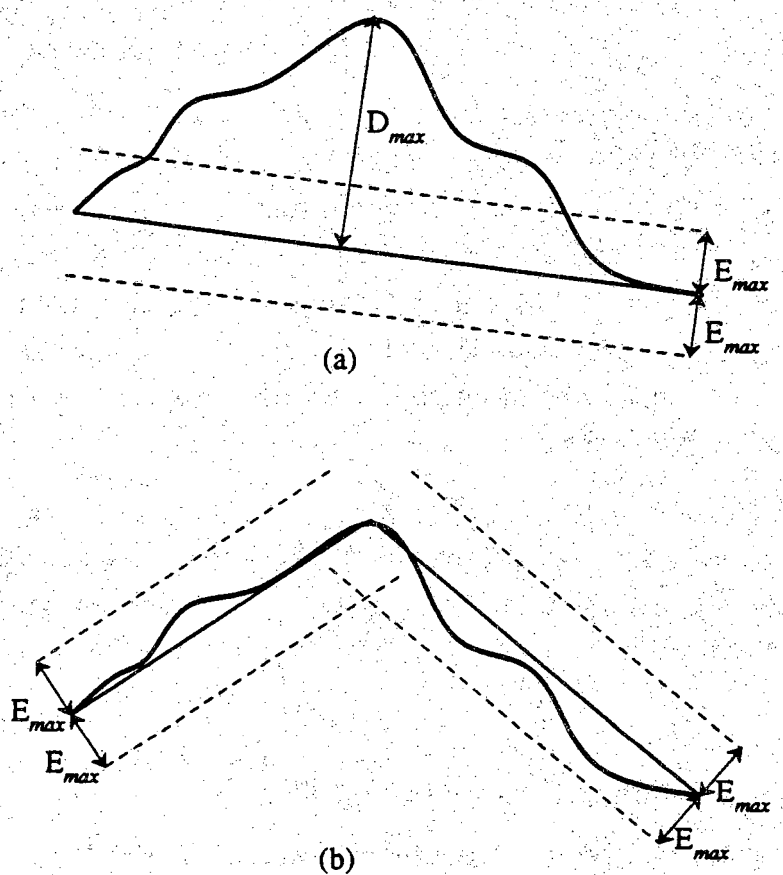


Figure 3.5 This figure demonstrates how a sample edge could be broken into two piecewise-linear segments by the line fitting algorithm. Since the edge falls outside the E_{max} boundaries in (a), the line is split into two in (b) where the edge lies within the E_{max} boundaries.

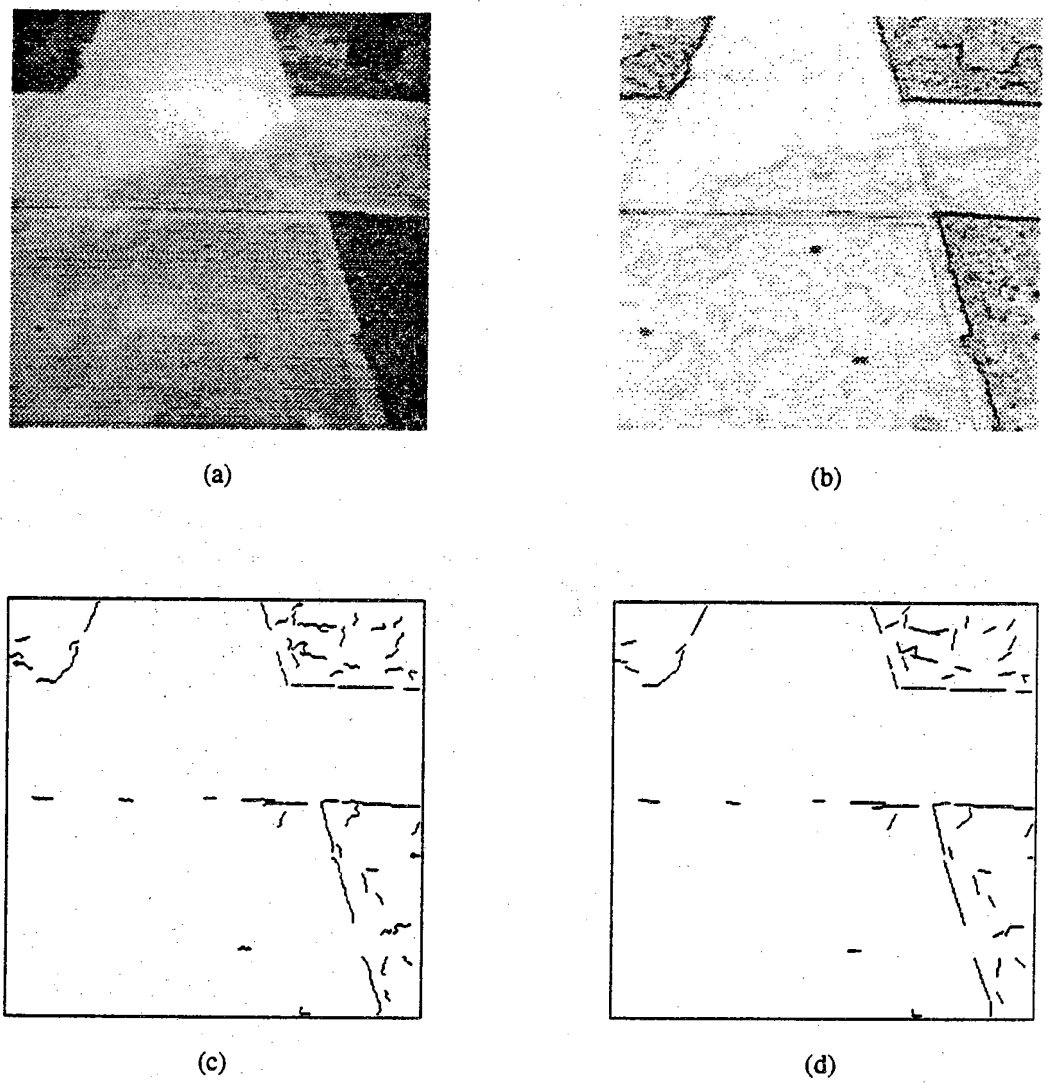
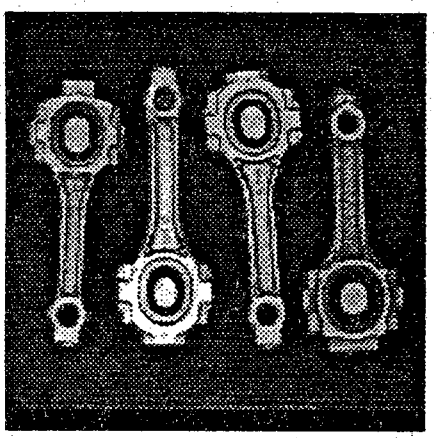
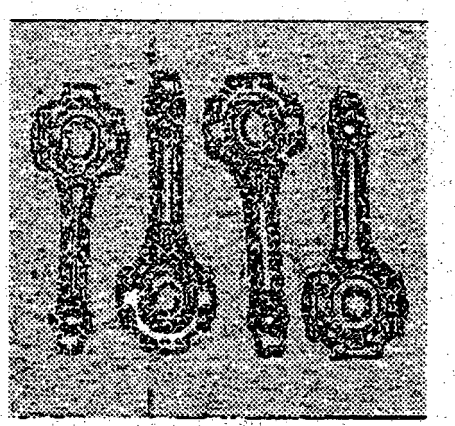


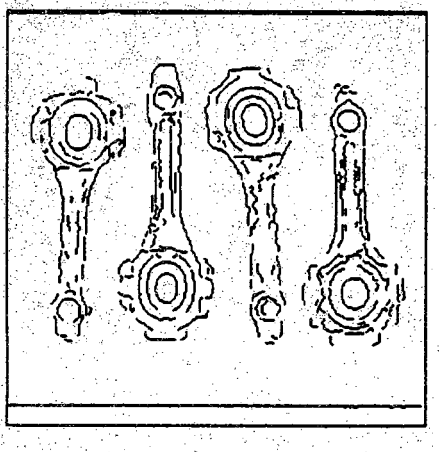
Figure 3.6 This figure shows the input, intermediate results and final output of the edge-based preprocessor when applied to an image typical of those gathered by a mobile robot with downward pointing cameras. Panel (a) shows the original image; panel (b) shows the image after applying the Sobel edge operator. Panel (c) shows the edges found by the ridge tracker and panel (d) shows the piecewise-linear edges output by the preprocessor.



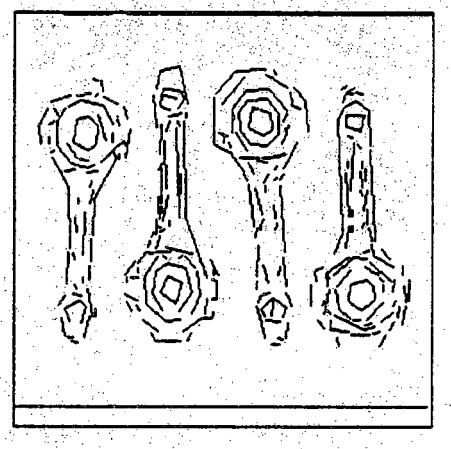
(a)



(b)



(c)



(d)

Figure 3.7 This figure shows the input, intermediate results and final output of the edge-based preprocessor when applied to an image typical of an industrial scene. Panel (a) shows the original image; panel (b) shows the image after applying the Sobel edge operator. Panel (c) shows the edges found by the ridge tracker and panel (d) shows the piecewise-linear edges output by the preprocessor.

1) The segmenter's first step is to convert the image into a data structure called a *greyscale quadtree*. A greyscale quadtree is a simple extension of the binary quadtree in which every leaf is maximal and satisfies a constraint (a leaf is maximal if it is not part of a larger leaf that satisfies the constraint). In this preprocessor, a group of pixels is allowed to be grouped into a leaf of a quadtree if

$$\left\{ \max_{x,\,y} f(x, y) - \min_{x,\,y} f(x, y) \right\} \leq 2\varepsilon \tag{3.1}$$

where f(x, y) denotes the brightness function of the image and x, y are allowed to range over the entire leaf; epsilon is a user-supplied parameter. In the original algorithm, this process required an iterative split-and-merge procedure. However, with the use of the Morton matrix [Mor66], [Sam84a] the quadtree can be built without any iterations. By visiting the pixels in the order defined by the Morton matrix (visit pixel 1 first, pixel 2 second, etc.), the building of a leaf can be postponed until it is known for certain that no larger leaf node satisfying constraint (3.1) is possible. For example, if the values of pixels 1-4 satisfy the constraint, but the values of pixels 1-5 do not, then the leaf defined by pixels 1-4 is guaranteed to be maximal. Thus, pixels 1-4 can be grouped into a leaf.

1	2	5	6	17	18	21	22
3	4	7	8	19	20	23	24
9	10	13	14	25	26	29	30
11	12	15	16	27	28	31	32
33	34	37	38	49	50	53	54
35	36	39	40	51	52	55	56
41	42	45	46	57	58	61	62
43	44	47	48	59	60	63	64

Figure 3.8 An example of an 8 by 8 Morton Matrix.

Furthermore, because of the geometry of quadtrees, the size of the leafs formed using pixels 5-16 can be at most 2×2. Fig. 3.8 shows an example of an 8×8 Morton matrix. Note that the Morton matrix does not have to be stored explicitly to guide the traversal of the image in the order that it prescribes. An image can be traversed in the correct order by recursively visiting the four quadrants of the image in the following order: upper-left, upper-right, lower-left, lower-right. For example, in an 8×8 image, the 4×4 quadrant in the upper left is visited first. Within this 4×4, the upper-left 2×2 subquadrant is visited first, etc.

The data structure shown in Fig. 3.9 is used to store the nodes in the greyscale quadtree. The *type* field is used as a flag to indicate whether or not the node is a leaf node. The *x*, *y* and *size* fields are used to specify the position of the upper-left corner of the node and its size (nodes are always square). The *links* field points to the children of a nonleaf node. If the node is a leaf node, then the links field points to the node's neighbors. The *region* field is used in later steps of the processing to indicate the region into which a node has been grouped. The final three fields are used to store statistics about the greyscale values of the pixels in the node. They store the minimum, maximum and average greyscale values. These three fields are used in later preprocessing steps to determine if adjacent nodes in the quadtree should be grouped together. If the image is not square or if its size is not an integral power of two, then the image is embedded in the northwest corner of the smallest quadtree that can contain it. In this case, the unused portion of the

```
struct node {
        int                 type;
        int                 x, y;
        int                 size;
        struct node     *links[4];
        struct node     *region;
        int                 min;
        int                 max;
        int                 average;
};
```

Figure 3.9 This data structure used to hold a node in the greyscale quadtree.

quadtree is indicated by NULL children links in nonleaf quadtree nodes. Panel (b) of Figs. 3.10 and 3.11 shows the leaves of the greyscale quadtree (the grey-levels in these panels are randomly generated to help the reader distinguish between adjacent regions).

2) The preprocessor's second step is to merge adjacent quadtree leaves into regions. Adjacent leaves are merged into a region only if the region formed also satisfies constraint (3.1). At the end of this step, each leaf node in the quadtree has been grouped into a region. The same data structure used to store quadtree nodes also is used to store regions; however, the *type* field is used as a flag indicating that the node is a region and the x, y and *links* fields are not used. It is possible to find the region into which a node has been grouped by following the *region* links in the data structure. The region links are used to form the tree based UNION-FIND data structure described in [AhoHop74]. Thus, region links do not necessarily point directly to a region node, they may point to a sub-region whose region link points to the region node. Thus, to merge two regions into a single larger region, the region link field of the smaller region is set to point to the larger region's node. When the linked list of region links is traversed to find a leaf node's region, the region links for all the nodes visited are set to point directly to the region node. Notice that the only way to find all of the nodes grouped into a specific region is to traverse the entire quadtree. If two adjacent nodes have been grouped into the same region, then the link information between them is reset to NULL. Panel (c) of Figs. 3.10 and 3.11 shows the regions of the image after the quadtree leaves are grouped based on constraint (3.1) (with randomly generated grey-levels).

3) In the third step in the process, adjacent regions whose average greyscale values differ by less then a user specified threshold are merged together. Panel (d) of Figs. 3.10 and 3.11 shows the regions of the image after this step has merged adjacent regions.

4) After these merging processes have been applied to the image, some small regions that should be eliminated may be left unmerged. For example, many of these regions are generated by shot noise and are only a single pixel large. This preprocessing step eliminates these small regions by merging all regions whose areas are less than a user specified value with the neighboring region whose average grey level is closest to its own. Panel (e) of Figs. 3.10 and 3.11 shows the regions of the image after the small regions have been eliminated.

5) The preprocessor's final step is to convert the preprocessed image into a format usable by PSEIKI. This is accomplished by first finding all the borders between regions; these border-elements then are converted into piecewise-liner segments using the process discussed in the previous section. The endpoint pixels are output as vertex-elements for the blackboard; likewise, the borders and regions are output as edges and faces respectively. Panel (f) of Figs. 3.10 and 3.11 shows the the borders between the regions after all of the substeps in step 5 have been applied. The conversion process is described below.

   5a) The first step of the conversion to PSEIKI's symbolic format consists of storing, in a set of linked lists, the borders between the regions. One list corresponds to each set of "cracks" between the rows and the columns in the original image; thus, for an M×N image, there are (M+1)+(N+1) border lists. Stored with each element of a list is the starting and ending position of the edge and pointers to the two regions that it borders. The borders are found by checking the four neighbors of each leaf node. If a leaf node and its neighbor are not in the same region, then the edge between them borders the two regions, and its information is stored in the appropriate border list.

   5b) After all of the borders are found, the border lists are searched for junctions of three or more borders. These junctions are used as the starting points of edges in the next preprocessing step.

   5c) The edges between the regions are formed by tracking the border segments stored in the border lists. The edges are tracked starting at a junction vertex; tracking stops as soon as another junction vertex is encountered. As each edge is tracked, its border segments are deleted from the border lists to prevent more than one edge from following the same border.

   5d) Finally, the procedure described in step 6 of the edge-based preprocessor is used to convert the edges into piecewise-linear segments. The region, edge and vertex data is then output in the format described in section 3.1.

This preprocessor is slightly less efficient than the edge-based system; it was able to preprocess 512×480 images in about two minutes on a SUN/3. It is currently believed that the face level information provided by this preprocessor justifies a slight decrease in efficiency.

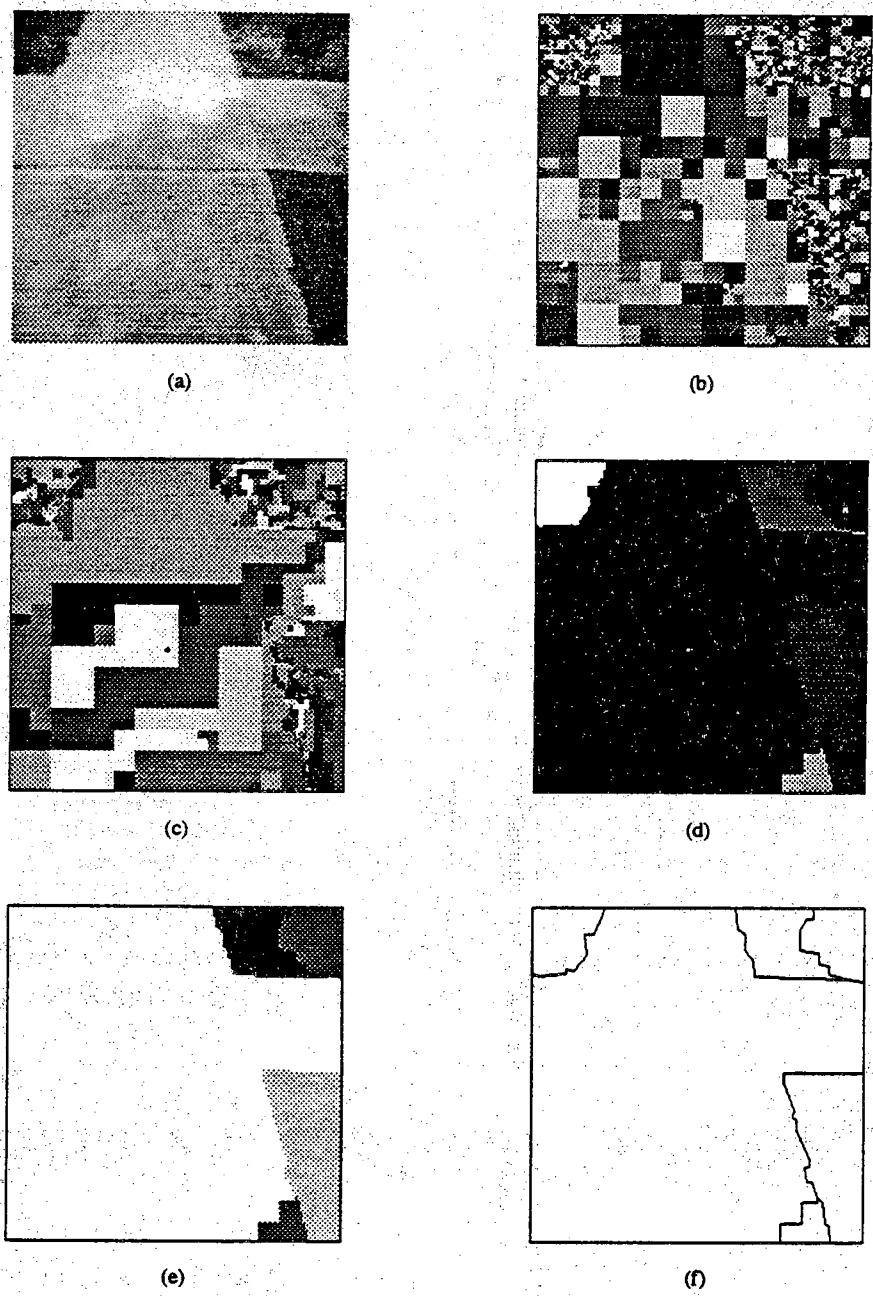


Figure 3.10 This figure shows the input image, intermediate results and final output of the region-based preprocessor when applied to an image typical of those gathered by a mobile robot with downward pointing cameras. Panel (a) shows the original image; panel (b) shows the leafs in the greyscale quadtree. Panel (c) shows the regions formed by the min/max merging; panel (d) shows the regions formed by the average value merging. Panel (e) shows the image after the small regions are eliminated and panel (f) shows the borders of the regions after they are converted into piecewise linear segments. The greyscale values for the regions in panels (b)-(e) are randomly generated to help the reader distinguish between adjacent regions.

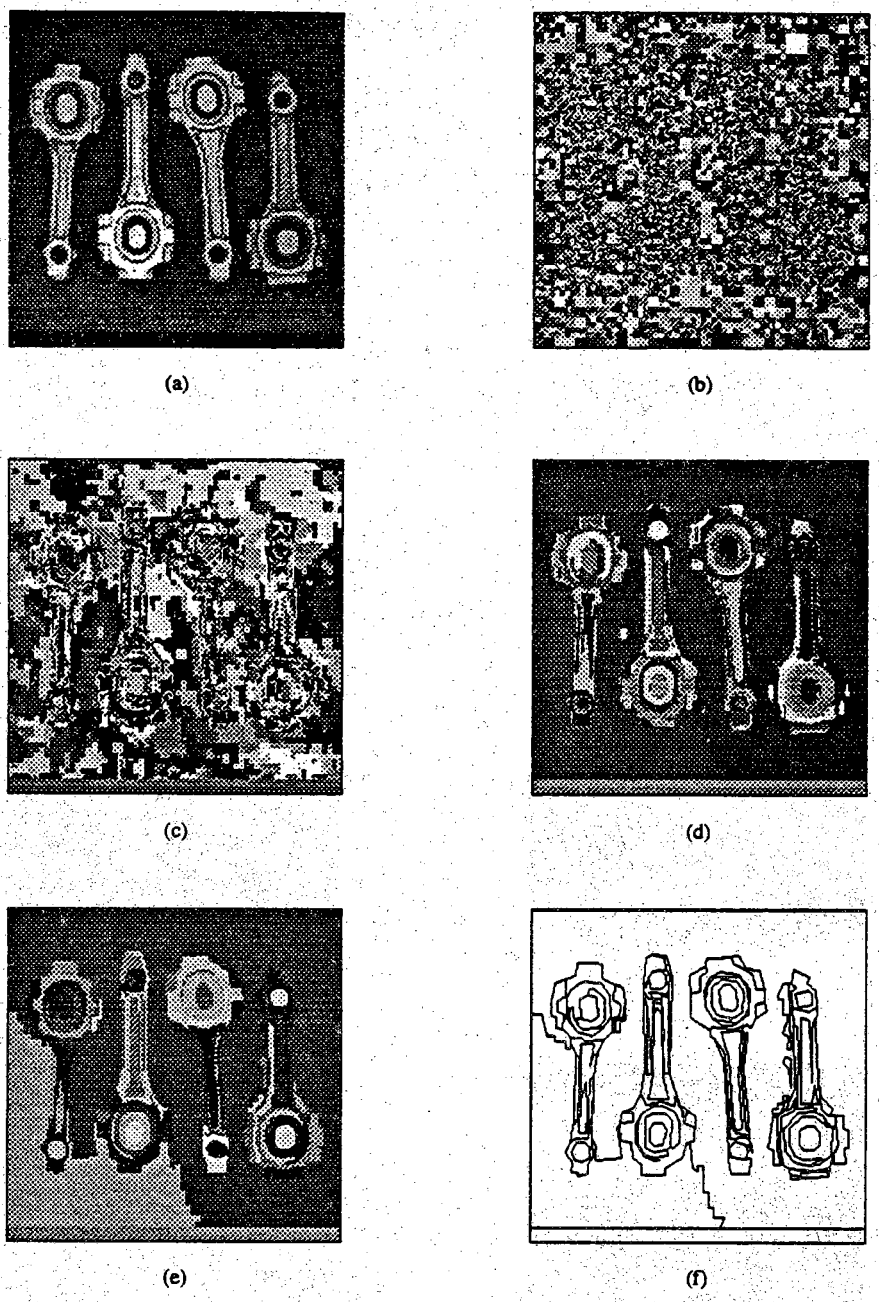


Figure 3.11 This figure shows the input image, intermediate results and final output of the region-based preprocessor when applied to an image typical of an industrial scene. Panel (a) shows the original image; panel (b) shows the leafs in the greyscale quadtree. Panel (c) shows the regions formed by the min/max merging; panel (d) shows the regions formed by the average value merging. Panel (e) shows the image after the small regions are eliminated and panel (f) shows the borders of the regions after they are converted into piecewise linear segments. The greyscale values for the regions in panels (b)-(e) are randomly generated to help the reader distinguish between adjacent regions.

CHAPTER 4

EXPECTED SCENE GENERATION

Computer graphics systems and CAD systems are two obvious methods of generating PSEIKI's expected scene information; this chapter will present two systems used to generate model information for PSEIKI. A computer graphics system is used to generate the expected scene information for a mobile robot traversing a network of sidewalks; the simple graphics-based generator can be employed because the sidewalk scenes are fundamentally 2-dimensional in nature. A solid modeling package is used to generate the expected scene in domains in which the scenes are 3-dimensional in nature. For example, the solid modeling system is used to generate the expected scenes when the mobile robot is indoors, traversing a network of building corridors. The same package could be used to generate the expected scenes in industrial domains. For example, it could be used if PSEIKI was employed as a verification vision system in a robotic assembly cell. Any modeling tool that is used for expected scene generation must possess the capability for hidden line removal. The modeling tool also must output its information in the format that was described in Section 3.1. Note that while the symbolic information input on the data panel initially has at most two or three levels, the expected scene has to be described as a hierarchy containing descriptions at all levels.

4.1. Expected Scene Generation for Sidewalk Navigation Applications

For sidewalk-navigation applications, a simple 2D graphics program is used to generate PSEIKI's expected scene information from stored sidewalk maps. In this system, the sidewalk maps are stored in a graph data structure. The links in this graph represent straight sections of the sidewalk and nodes represent the endpoints of the straight

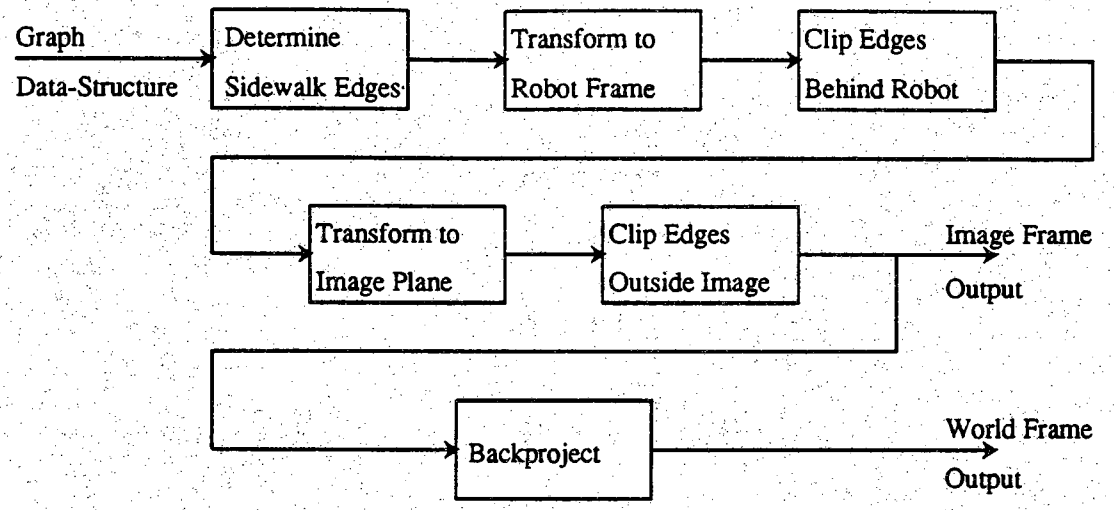


Figure 4.1 This figure shows a block diagram of the processes used to generate PSEIKI's expected scene information in a mobile robotic context.

sections. Associated with each node is an (x, y) pair designating the coordinates of the sidewalk junction corresponding to the node; thus, the centerline of a straight section of sidewalk is the line that connects the coordinates of its two junction nodes. Associated with each link of the graph is a value specifying the width of the corresponding sidewalk segment. This information completely specifies a sidewalk map.

Fig. 4.1 illustrates the steps involved in the generation of a symbolic description of the expected scene from the graph data structure. The first step is the extraction of the edges of the sidewalk from the graph data structure. It is a trivial task to determine the lines defining the edges of a straight section of the sidewalk because both the section's width and its centerline are known. A more difficult problem is encountered when trying to determine the location of the vertices corresponding to the intersection points of the edges of the sidewalk.

The following algorithm is used to determine the location of the vertices. First, four vertices are associated with each link in the graph; the vertices correspond to the two endpoints of each of the sidewalk segment's two edges. For example, the vertices P, Q, R and S are associated with the link AB as shown in Fig. 4.2. Vertices P and Q are obtained by analyzing node B, whereas vertices R and S are obtained by analyzing node

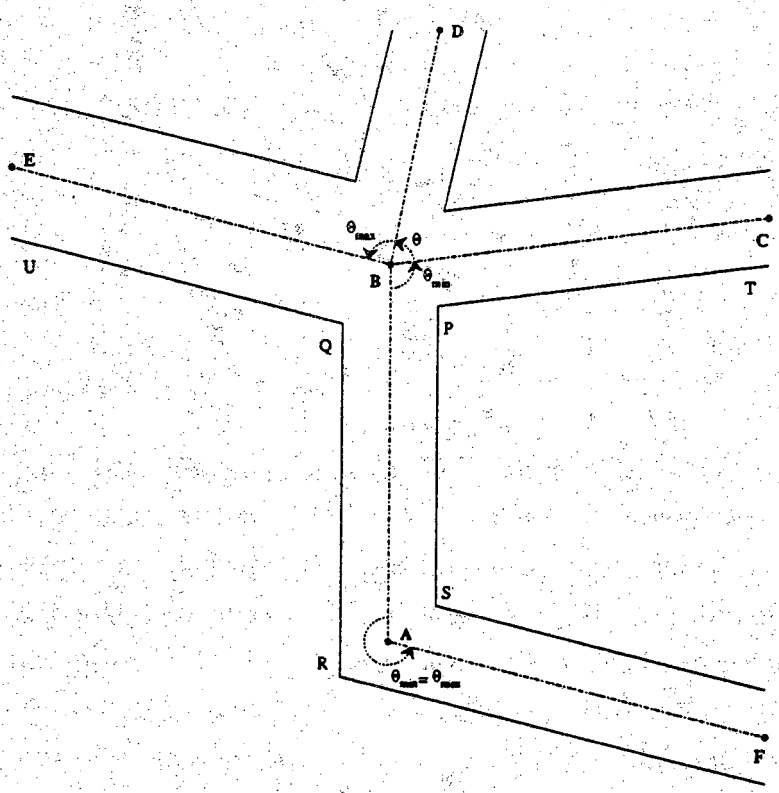


Figure 4.2 This figure shows a part of a sidewalk map; it shows the process used to generate a sidewalk's edges from a graph description.

A. Consider node B first. The graph is searched for all the links that meet at B; then the angle between each link and link AB is then calculated (all angles are measured in a counterclockwise direction). Then only those links that correspond to the minimum and maximum of these angles are retained. In the figure, links BC and BE correspond to the minimum and maximum angles, respectively. Now it is a simple matter to compute the location of the two vertices, P and Q, that correspond to node B of link AB. The computation of the location of vertex P can be found by solving the equations of the straight lines corresponding to the edges SP and PT. Similarly, the location of vertices R and S can be computed by analyzing node A. At a node where there is a bend in the sidewalk, as opposed to a junction, the minimum and the maximum angles correspond to the same link. For example, at node A, the minimum and the maximum angles both correspond to the same link, link AF. The algorithm is presented as pseudo-code in Fig. 4.3. The

```
/*
 * these routines find the vertices of both edges of
 * a sidewalk link.
 */
get_both_edges(LINK, r_start, r_end, l_start, l_end) {
    get_vertices(LINK, start_node(LINK), l_start, r_start)
    get_vertices(LINK, end_node(LINK), r_end, l_end)
}

/*
 * this routine finds the vertices of one edge of a sidewalk link
 */
get_vertices(LINK, NODE, r_vertex, l_vertex) {
    for each link in the graph not equal to LINK {
        if (one of the link's nodes is equal to NODE)
            add the link to the set of intersecting links
    }

    /*
     * sort the intersecting links on the basis of
     * the angle between them and LINK
     */
    min_link = link with minimum angle
    max_link = link with maximum angle

    r_vertex = intersect(edge(LINK, right), edge(min_link, right))
    l_vertex = intersect(edge(LINK, left), edge(max_link, left))
}
```

Figure 4.3 This figure shows the pseudo code for algorithm used to determine the location of the vertices of a section of the sidewalk.

reader should note that this procedure will yield each vertex twice. In this example, the vertex corresponding to point P will be generated when node B is considered as belonging to link AB, and then again when the same node is considered as belonging to link BC. This duplication of the symbolic description at the vertex level is easily eliminated by comparing vertices and dropping one from each pair found to have nearly identical coordinates.

After a symbolic description of the edges in the sidewalk map has been extracted from the graph data structure, a "spotlight" function is applied to the description to delete all those edges that are not visible given the robot's hypothesized location and orientation. To implement the spotlight function, two homogeneous transformation matrices are generated, one that takes a world point into the robot base coordinate frame

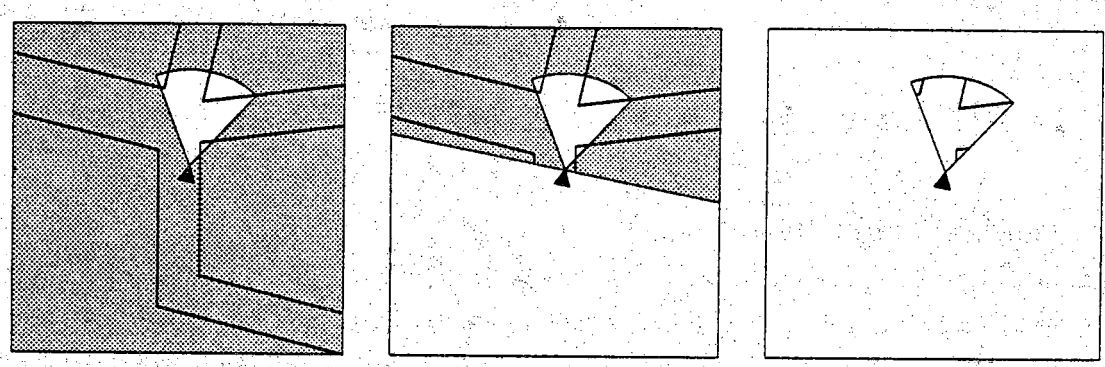


Figure 4.4 This figure shows how the spotlight function is used to delete from the expected scene all edges that can not be seen from the robot's hypothesized location and orientation. In the leftmost panel, the triangle shows the expected location and orientation of the robot and the unshaded area shows the region of the ground visible to the robot's downward slanted cameras. The center panel shows the clipping of the edges behind the robot. The rightmost panel shows the edges remaining after the image-coordinate clipping is performed.

and the other that takes a point from the robot base coordinate frame into the camera image plane. The first matrix is derived from knowledge of robot's location and orientation; it is used to transform end points of edges, such as point P for edge PS in Fig. 4.2, from the world frame into a robot base coordinate frame. A clipping operator is applied to the transformed data to delete all edges that are behind the robot. The middle panel of Fig. 4.4 illustrates the edges from the left panel that would remain after this clipping operation is applied. The second transformation matrix, which is derived from camera calibration parameters, is used to project the clipped edges onto the camera image plane. Finally, a second clipping algorithm is applied to delete the edges and parts of the edges that fall outside the boundaries of the image. The edges from the middle panel of Fig. 4.4 that are not deleted by the final clipping operation are shown in the right panel. Note that the edges of the sidewalk are still described symbolically at this point; that is, they have not been converted into image form.

If the expected scene is to be expressed in image coordinates, the vertex and edge level information is output in the format described in section 3.1. If the expected scene is to be expressed in world coordinates, the clipped edges are back-projected into the world coordinate frame and output in the appropriate format. The project/clip/back-project

process just described has the desired effect of deleting all edges that are not visible from the robot's hypothesized location and orientation. Although it would be possible to implement the world coordinate spotlight function via a simple clipping operation performed in the world coordinate frame, using the project/clip/back-project algorithm allows a single spotlight function to be used for both world coordinate and image coordinate output.

After the low-level information is generated by the graphics system, the model information on the face level, the object level and the scene level is hand entered by editing the output file. On the face level, each connected section of sidewalk and each connected section of the ground is considered to be a separate face. These faces are hand grouped into the single object in the scene. To help the operator enter this upper-level information, an image of the expected scene, with the grey values of each edge indicating its symbolic id number, is displayed at the same time the low-level symbolic output is generated. Generating this image is trivial because the spotlight function projects the sidewalk's edges into the image coordinate plane. Hand entering the upper-level information usually is not difficult because the sharp down-look angle of the camera limits the complexity of the expected scenes.

As an example of the processing performed by this graphics system, consider the following figures: Fig. 4.5 shows a simple sidewalk map to be used in this example. Fig. 4.6 shows a sequence of expected-scene images that the system would produce for a mobile robot traveling to the middle sidewalk section of the map, turning up that section and then turning right.

4.2. Expected Scene Generation for Indoor Navigation Applications

A solid modeling system is used to generate PSEIKI's expected-scene data for mobile robotic applications in which the robot is indoors, traversing a network of building corridors. The solid modeling information is required because the robot's environment, the building's hallways, contains a great deal of 3-dimensional information. Because the modeling system can represent a wide range of solids, the solid modeling system could be used for other applications with expected scenes containing large amounts of 3-dimensional information. In the past decade, solid modeling techniques have gained great popularity for representing geometric objects in a complete and unambiguous fashion. Constructive solid geometry (CSG) and boundary-representation (B-

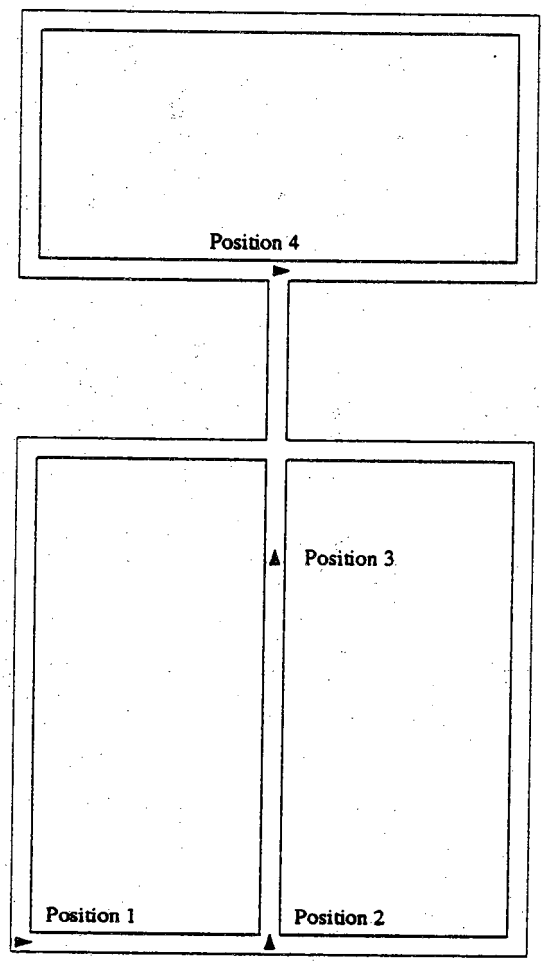


Figure 4.5 This figure shows the sidewalk map used to generate the expected scene images of figure 4.6. The robot's position for each of the four expected scenes is indicated in the drawing.

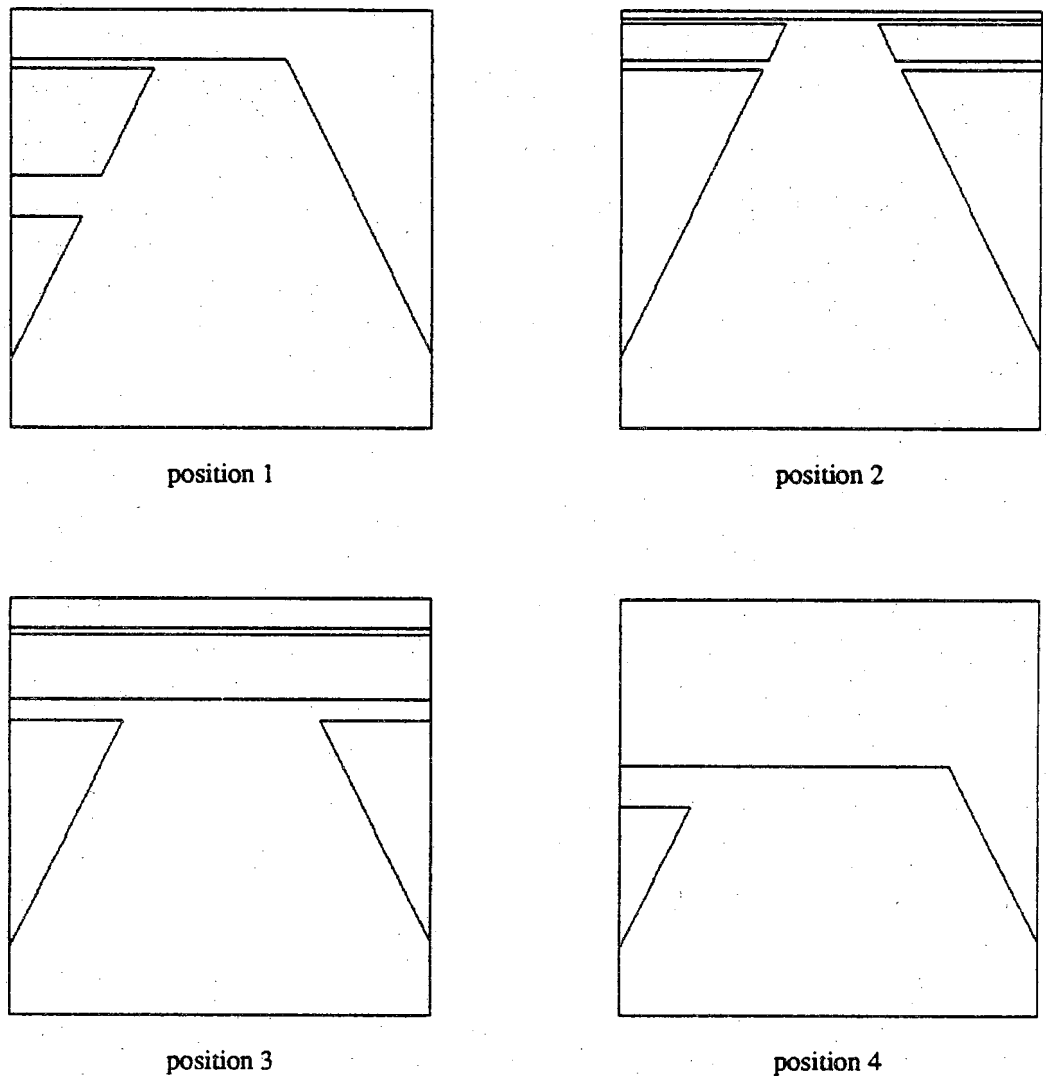


Figure 4.6 This figure shows some typical expected scenes generated for a mobile robot with downward pointing cameras. The scenes depicted in this figure were generated with the map shown in Fig. 4.5.

rep) are two popular solid-modeling techniques. In this section, both CSG and B-rep modeling principles will be described. The TWIN B-rep solid modeling system currently being used to generate PSEIKI's expected scenes then will be described in this context. Finally, two example uses of the expected scene generator will be given. First, an example of the generation of hallway expected scenes will be shown. Second, the use of the modeler to generate expected scenes in industrial domains also will be briefly explored.

CSG systems combine primitive objects into arbitrary solid objects using the following boolean operators: union, intersection and difference. Fig. 4.7 shows how a simple object can be constructed by combining primitive solids using these CSG operators. CSG systems usually are restricted to working with *regular* solids. A set of points, X, is said to be regular if it is equal to the closure of its interior, that is

$$X = ki(X)$$

where k and i denote the closure and interior, respectively. Because a solid produced by the combination of regular solids using the set-theoretic boolean operations is not necessarily regular, To guarantee that the result of a combination will be regular, CSG systems use *regularized* boolean operators when combining objects. Fig 4.8 shows how a non-regular object can result from the set-theoretic intersection of two regular objects; it also shows the regularized intersection of the two objects. The set-theoretic intersection of the two faces in panel (a) of Fig. 4.8 is shown in panel (b); note that the result of the combination is not regular (because of the "dangling" edge). Panel (c) shows the valid face produced by taking the regularized intersection of the two faces in panel (a). The set-theoretic union and difference operators have similar problems. The regularized operators, union ($\cup^*$), intersection ($\cap^*$) and difference ($-^*$), of two sets, X and Y, are defined as

$$X \cup^* Y = ki(X \cup Y)$$

$$X \cap^* Y = ki(X \cap Y)$$

$$X -^* Y = ki(X - Y)$$

Most of the concepts used in CSG modeling systems were originally developed for the PADL solid modeling system [VoeReq77], [HarMar85].

Boundary-representation modeling is another common solid-modeling technique. In this scheme, objects are represented in terms of their boundary surfaces. In many B-rep systems, polyhedrons are used to approximate the boundary of the objects; thus, any

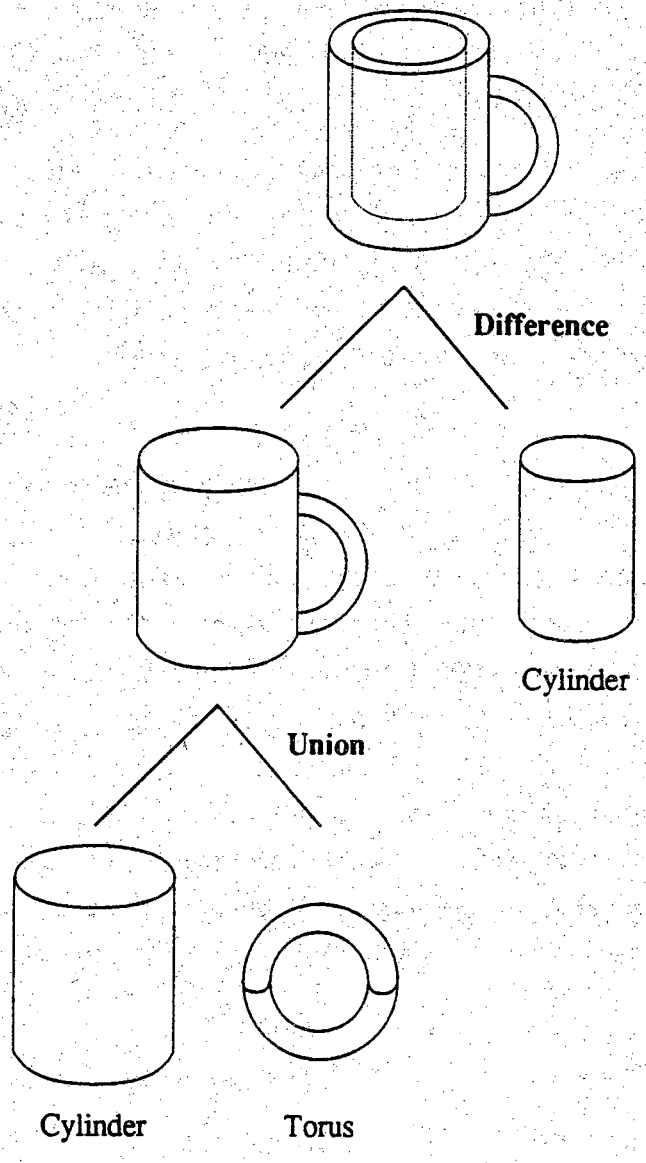


Figure 4.7 This figure demonstrates how objects are defined in CSG systems by the boolean combination of successively simpler objects. The coffee mug in this figure is defined in terms of two cylindrical primitives and one toroidal primitive.

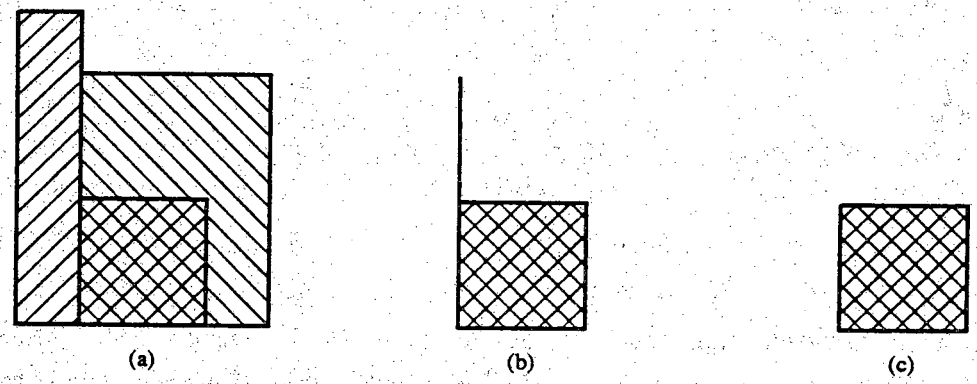


Figure 4.8 This figure shows a shortcoming of the set-theoretic intersection operation in two dimensions. The face in panel (b) is the set-theoretic intersection of the two faces in panel (a); it is not a valid face (because of the dangling edge). The face in panel (c), a valid face, is the regularized intersection of the two faces.

curved surfaces, such as cylindrical or spherical surfaces, are represented only approximately. PSEIKI uses the TWIN B-rep solid modeling package [Mas87] to generate expected scene information in an industrial domain. TWIN was developed at the Computer Aided Design and Graphics Laboratory (CADLAB) at Purdue University's Engineering Research Center. TWIN is a library of C language subroutines that contains routines to generate the primitive objects included in most CSG systems; the set of primitives that TWIN can generate includes parallelepipeds, wedges, cylinders, cones, toruses, spheres, fillets, elliptical cones, and ellipsoids. The library also contains routines to perform regularized boolean operations on solid objects. Because the TWIN library contains routines to generate the primitives used in CSG systems and routines to perform the operations used by CSG systems, the process used to generate solid objects in CSG systems also can be used to generate objects with TWIN. That is, solid objects can be defined by regularized boolean combinations of primitive objects.

A three step procedure is used to convert a TWIN model into a form usable by PSEIKI. First, the scene is intersected with a halfspace such that all scene elements behind the image plane are removed. This first step is needed because the TWIN rendering algorithm used in the second step produces errors if there are any objects extending behind the camera. In the hallway navigation application, the camera is located inside the hallway and therefore needs to have the surfaces behind the camera deleted. In

industrial applications where the objects are guaranteed to be located in front of the image plane, this first step is not needed. Second, the Watkins scan-line rendering algorithm [Wat70] is used to generate an image of the expected scene. The grey value of every pixel in the rendered image is set to the ID number of the model surface visible at that location in the image; thus, all regions corresponding to the same surface in the TWIN model have the same grey level in the image. Third, after the model is converted into an image, the region-based preprocessor described in chapter 3 is used to extract the image's labeled regions and output the scene description on the vertex, edge and face levels. The threshold values required by the segmenter are set to zero so that each region detected by the segmenter will correspond to a single model surface. The information on the object and scene levels is generated by assuming that only a single object is present in the expected scene. Thus, all of the regions detected in the image, with the exception of the background region (which has id number zero), are grouped into a single object. Then, this object is set to be the only object in the scene. If there is more than one object in the image, then the output file must be hand edited to correct the object and scene level information. Note that using an image segmentation process to generate the expected-scene results in a face level partition of the scene; that is, no faces overlap.

Fig 4.9 shows an overhead diagram of a hallway in Purdue's Electrical Engineering Building. Fig 4.10 shows four expected scenes produced by a camera located at the arrows shown in the diagram in Fig. 4.9. Fig. 4.11 demonstrates that the expected-scene generator can also be used to generate expected scenes in an industrial application. This figure shows a graphic output of the system for an industrial object, a piston connecting rod; this figure shows three orthogonal views and one oblique view of the object. Fig 4.12 illustrates the process used to generate the expected-scene information for the hallway scene shown in Fig 4.10(a). The image at the top of the figure represents the TWIN solid model. The image at the middle of the figure shows the rendered image with uniquely labeled surfaces. A small portion of the symbolic output is shown at the bottom of the figure. In reality, this data file contains the definitions for about 200 elements.

The current method of generating PSEIKI's expected scene information has an obvious flaw. The main deficiency of the technique is the assumption that there is only one object visible in the expected scene. Because the faces for all of the objects in the scene are grouped together, each face will be used to update the belief in every other face, regardless of the object to which the face belongs. For example, assume an expected scene for an indoor navigation experiment contained two objects: the hallway

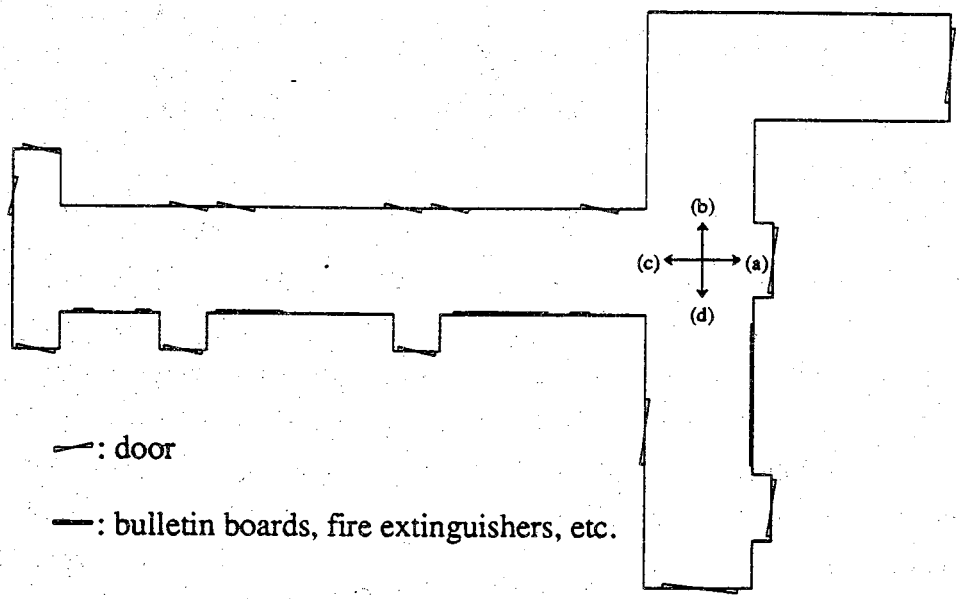


Figure 4.9 This figure shows a diagram of the hallway used as a model for the expected scenes shown in Fig 4.10.

walls and a file cabinet. If the file cabinet in the hallway was displaced from its expected position, then the relationship between cabinet's faces and the faces representing the hallway's walls would not mimic their relationships exhibited in the expected scene. Thus disconfirmatory evidence would be generated for the correct identity of all the faces in the scene. However, if the faces were correctly grouped into multiple objects, then the faces from both of the objects would be determined to be consistent (because the geometric relationships of the faces composing the hallway and the cabinet, considered individually, do not depend on the position of cabinet in the hallway). Currently, if there is more than one object in the scene, then the upper-level information must be corrected by hand. It usually is not difficult to hand correct this information; however, it would be convenient if the system was able to generate PSEIKI's input data correctly at all levels of abstraction.

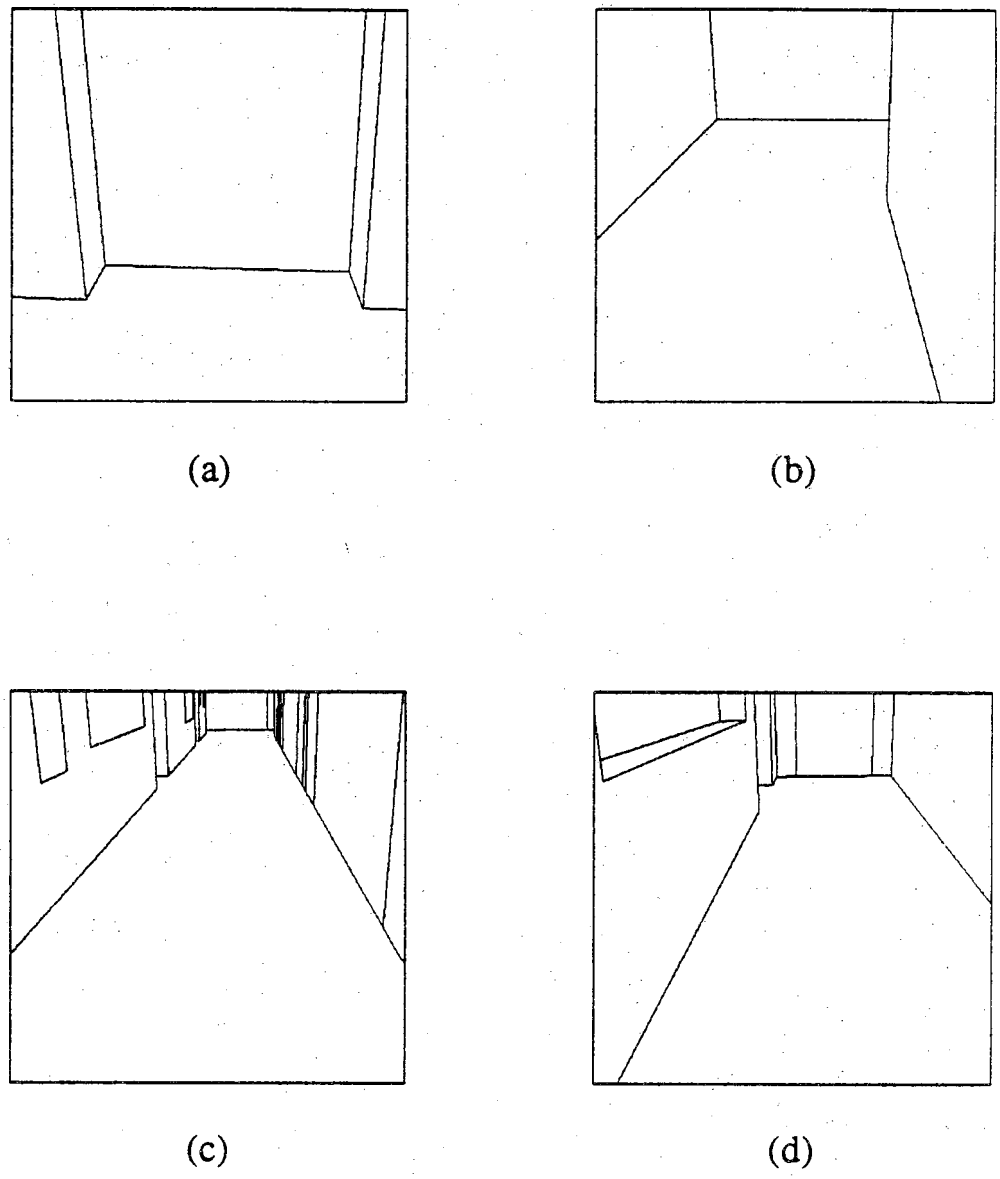


Figure 4.10 This figure shows four expected scenes produced by a camera located at the arrows shown in the diagram in Fig. 4.9.

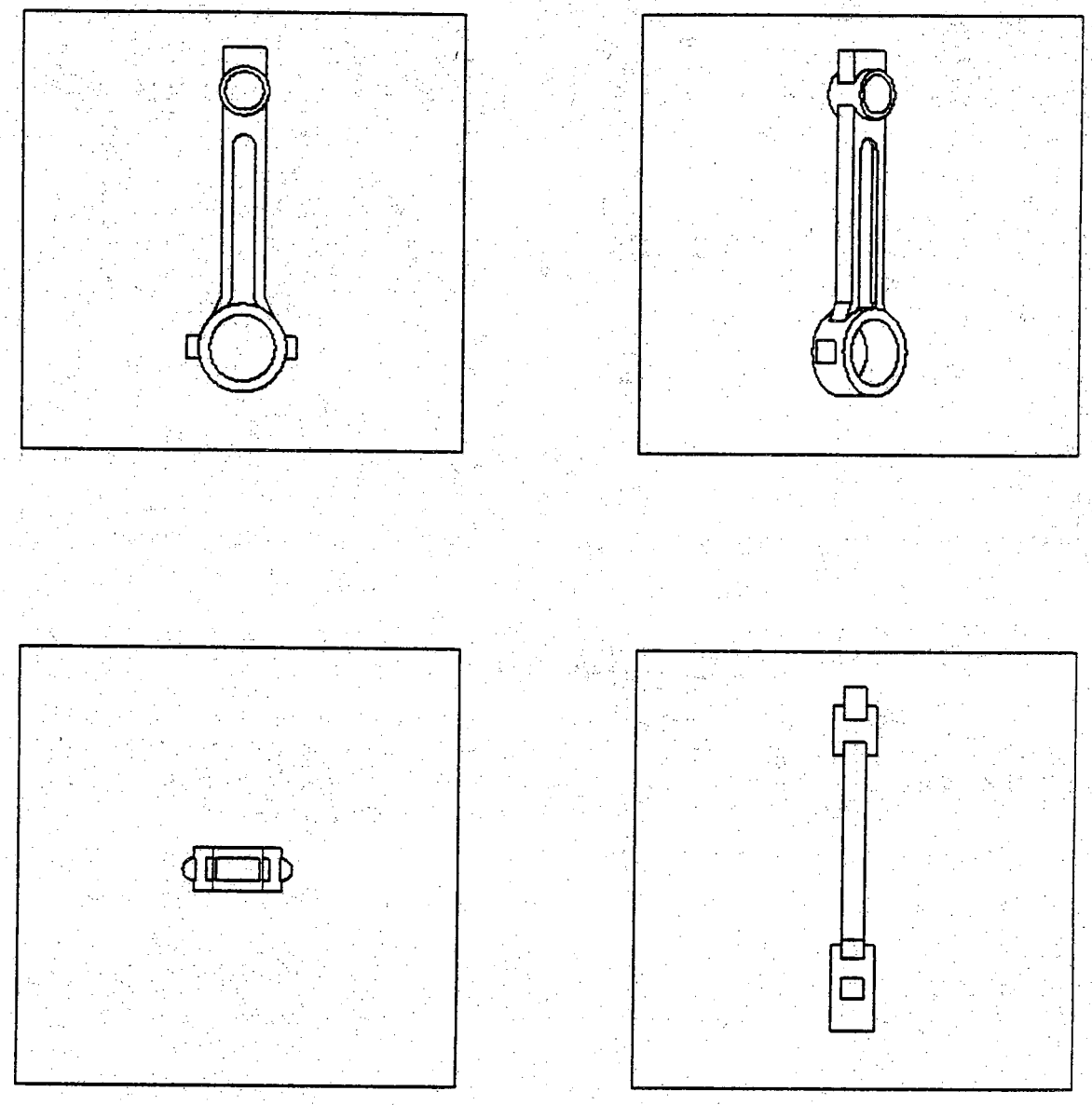
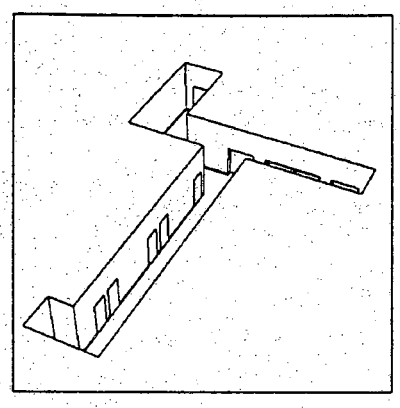
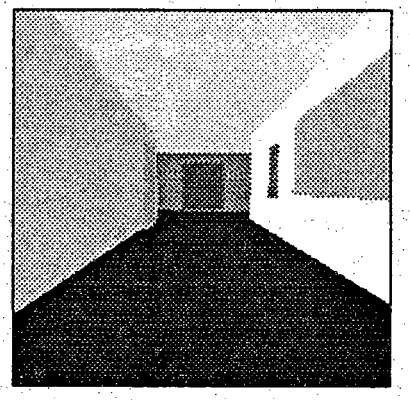


Figure 4.11 This figure shows some typical expected scenes generated in an industrial environment. It shows a piston rod from three orthogonal views and one perspective view.



TWIN solid model

Watkins rendering algorithm



Rendered image

Region-based Segmentation

+ scene id 2194 children 2193
+ object id 2193 children 2174 2175 2176 2177 2178 2179 2180 2181 2182 2183 ...
+ face id 2174 children 2149 2161 2162 2163 2164 size 684
+ face id 2175 children 2103 2145 2146 2147 2148 2149 size 924
+ face id 2176 children 2106 2107 2108 2109 2142 2143 2145 2146 2155 2156 ...
+ face id 2177 children 2084 2085 2086 2087 2088 2089 2090 2091 size 1098
⋮

Figure 4.12 This figure shows the processing performed to generate the symbolic expected scene data from solid models. The top image represents the TWIN solid model information; the middle panel shows the rendered model image with every surface uniquely labeled. The lower part of the image shows a small portion of the symbolic output which would be presented to PSEIKI.

CHAPTER 5

AN EVIDENCE ACCUMULATION SCHEME FOR BLACKBOARD REASONING

The use of inexact reasoning in computer vision systems is certainly not new; however, most of the previous schemes for evidence accumulation have been based only loosely on formal uncertainty calculi [HanRis78], [MckHar85]. The main reason that these systems employed ad-hoc schemes is the overwhelming amount of data in an image; the systems needed a fairly simplistic evidence accumulation scheme to avoid becoming bogged down in confidence value computations. In contrast, the evidence accumulation scheme used in PSEIKI is based on the Dempster-Shafer (D-S) theory of evidence whose usual exponential computational complexity is controlled by a number of mechanisms to be discussed in this chapter. For example, one of the mechanisms consists of limiting evidence to one of two types; it either confirms or denies the proposition that an image element should be matched with a particular model element. Pooling of evidence in this fashion leads to a particularly efficient implementation of the Dempster's rule. Performance also is improved by exploiting the hierarchical nature of the blackboard system. By performing a small number of computations on upper levels of the hierarchy, many computations on lower levels can be avoided. The hierarchical nature of the blackboard also is used to constrain the matching process for elements on lower levels of the hierarchy; elements on lower levels of the hierarchy are allowed to match only if their parents are matched.

. In the next section of this chapter, Dempster's rule of combination will be introduced, and its exponential time complexity will be noted. Next, it will be shown how the computational efficiency of Dempster's rule can be improved by assuming that the focus of incoming evidence is limited to a small number of subsets of Θ. Once these

assumptions are made, computationally efficient forms of Dempster's rule can be derived. One such assumption states that all evidence either confirms or denys individual elements in the FOD. Using this assumption, Barnett [Bar81] was able to implement Dempster's equation in linear time. The evidence accumulation scheme employed by PSEIKI is based on Barnett's linear implementation; the new accumulation scheme will be introduced first using Barnett's framework. Next, it will be shown that the accumulation scheme can be embedded into a hierarchy if the reasoning task has the appropriate structure. It also will be shown that the hierarchical structure allows the computational complexity of the scheme to be improved by limiting the size of the elements' FODs and by limiting the number of evidence sources that are allowed to provide evidence. Furthermore, a method for passing belief values up the hierarchy will be introduced. After the general scheme has been fully developed, its use by PSEIKI's labeler KS will be presented as an application. Finally, to show the generality of the new evidence accumulation scheme, its application to the speech recognition domain will be noted briefly.

5.1. A Brief Review of the Dempster-Shafer Theory of Evidence

The Dempster-Shafer (D-S) theory of evidence is gaining wider acceptance as an uncertainty calculus. In this section, a short review of some relevant terminology from the D-S theory of evidence accumulation will be presented. For a detailed presentation of the theory, the reader is referred to Shafer [Sha76].

In a random experiment, the *frame of discernment* (FOD), Θ, is the set of all possible outcomes. For example, if a die is rolled, Θ can represent the set of possibilities, "the number showing is i," where $1 \leq i \leq 6$; therefore, Θ may be defined as the set $\{1, 2, 3, 4, 5, 6\}$. The $2^{|\Theta|}$ subsets of Θ are called propositions and the set of all the propositions is denoted by 2^{Θ}. In the die example, the proposition "the number showing is even" would be represented by the set $\{2, 4, 6\}$. The members of a FOD are known as *singleton propositions* or merely *singletons*.

In the D-S theory, *probability masses* are assigned to propositions, (i.e. to the subsets of Θ). This is a major departure from the Bayesian formalism in which probability masses must be assigned to the singleton propositions of Θ. The probability masses assigned to a subset, ψ, of the FOD is a measure of the total belief committed exactly to ψ. This belief cannot be further subdivided among the subsets of ψ and does not include the measures of belief committed to the subsets of ψ. Thus, the mass assigned to ψ is

constrained to stay within the subset but is free to move to any element of the subset. The probability mass assigned to the FOD represents ignorance because the mass assigned to Θ may move to any element of the entire FOD. The probability masses assigned to the propositions must have unity sum. When a source of evidence assigns probability masses to the propositions discerned by Θ, the resulting function is called a *basic probability assignment* (BPA). Formally, a BPA is function $m:2^{\Theta} \rightarrow [0,1]$ where

$$\begin{aligned} &1)\ m(\varnothing) = 0 \\ &2)\ 0.0 \le m(\cdot) \le 1.0 \\ &3)\ \sum_{\psi \subseteq \Theta} m(\psi) = 1.0 \end{aligned} \tag{5.1}$$

For example, assume that there is evidence that an even number is showing on a die with degree 0.5 and there is evidence that the number showing is two with degree 0.4. Then the remaining belief, $1 - 0.5 - 0.4 = 0.1$, is assigned to Θ.

$$\begin{aligned} m(\{2, 4, 6\}) &= 0.5 \\ m(\{2\}) &= 0.4 \\ m(\Theta) &= 0.1 \end{aligned}$$

A subset, ψ, of Θ is called a focal element of the belief function if $m(\psi) \neq 0$.

A belief function, $Bel(\psi)$, over Θ is defined by

$$Bel(\psi) = \sum_{Y \subseteq \psi} m(Y) \tag{5.2}$$

In other words, the belief in a proposition ψ is the sum of probability masses assigned to all the propositions implied by ψ. Thus, $Bel(\psi)$ is the measure of the belief in all subsets of ψ, and not the amount allocated precisely to ψ. Note that the belief in any singleton is equal to its probability mass. For example, using the above BPA, the belief that an even number is showing is

$$Bel(\{2, 4, 6\}) = m(\{2, 4, 6\}) + m(\{2\}) = 0.5 + 0.4 = 0.9$$

Dempster's rule of combination, states that two BPA's, $m_1(\cdot)$ and $m_2(\cdot)$, corresponding to two independent sources of evidence†, may be combined to yield a new

† The independence of PSEIKI's evidence sources will be discussed in chapter 6.

BPA m(·) via

$$m(\psi) = K \sum_{\substack{\psi_1 \; \psi_2 \\ \psi_1 \cap \psi_2 = \psi}} \sum m_1(\psi_1) m_2(\psi_2) \tag{5.3}$$

where K does not depend on ψ.

$$K^{-1} = 1 - \sum_{\substack{\psi_1 \; \psi_2 \\ \psi_1 \cap \psi_2 = \varnothing}} \sum m_1(\psi_1) m_2(\psi_2)$$

This formula is commonly called *Dempster's rule* or *Dempster's orthogonal sum* and is denoted as

$$m = m_1 \oplus m_2$$

If K^{-1} is equal to zero, then the two input belief functions are said to be *completely contradictory* and the result of the combination is not defined.

In the general case, Dempster's sum takes exponential time (in the size of the FOD) to combine evidence from two independent sources. This is shown easily by observing the formula for Dempster's sum as shown in equation (5.3). The main reason for the exponential complexity is the requirement that the probability mass for all $2^{|\Theta|}$ subsets of Θ be evaluated when combining evidence from independent sources. In the general case, it also is necessary to enumerate all $2^{|\Theta|}$ subsets of Θ when computing the belief of an arbitrary proposition from the BPA. If N BPA's are combined to form a data-element's belief function, then the total number of operations will be on the order of $N \times 2^{|\Theta|}$ (this will be denoted as $O(N \times 2^{|\Theta|})$).

There are a number of special types of belief functions. A belief function with at most one focal element (not counting the entire FOD, Θ) is called a *simple support function*. A *separable support function* is either a simple support function or is equal to the orthogonal sum of two or more simple support functions. *Dichotomous belief functions* [ShaLog87] are belief functions with focal elements $\{\psi, \neg\psi, \Theta\}$ for some subset ψ of Θ. The BPAs from two dichotomous belief functions with identical focal elements, $\{\psi, \neg\psi, \Theta\}$ can be combined in constant time. Equation (5.4) shows the special case of Dempster's rule that can be used to combine two dichotomous belief functions without enumerating all of the subsets of Θ. Dichotomous belief functions are used extensively in PSEIKI to accumulate evidence efficiently.

$$m(\psi) = \frac{m_1(\psi)m_2(\psi) + m_1(\psi)m_2(\Theta) + m_1(\Theta)m_2(\psi)}{1-m_1(\psi)m_2(\neg\psi)} \tag{5.4}$$

$$m(\neg\psi) = \frac{m_1(\neg\psi)m_2(\neg\psi) + m_1(\neg\psi)m_2(\Theta) + m_1(\Theta)m_2(\neg\psi)}{1-m_1(\psi)m_2(\neg\psi)}$$

$$m(\Theta) = \frac{m_1(\Theta)m_2(\Theta)}{1-m_1(\psi)m_2(\neg\psi)}$$

5.2. Computationally Feasible Methods For Evidence Accumulation Based on the Dempster-Shafer Theory

5.2.1. Barnett's Implementation of Dempster's Rule in Linear Time

Barnett [Bar81] was one of the first to show that Dempster's rule could be implemented in better than exponential time if the focus for all evidence is restricted to a limited number of subsets of Θ. Barnett was able to implement Dempster's rule using a linear time algorithm by assuming that all evidence either confirms or denies members of the FOD. Thus the belief functions being combined are all simple support functions focused on singletons or their complements. Although this assumption places a fairly large restriction on the general D-S theory, many systems naturally provide evidence in this form and are not hindered by the assumption.

Without loss of generality, let Θ be a FOD with n elements, $\Theta = \{i \mid 1 \le i \le n\}$. For each $i \in \Theta$, assume that the j^{th} simple support function with focus $\{i\}$ is denoted as $m_j^i(\cdot)$. Also assume that the j^{th} simple support function with focus $\neg\{i\}$ is denoted as $m_j^{\neg i}(\cdot)$. Barnett describes a three step procedure to combine simple support functions of this form into a single separable belief function.

1) First, for each $i \in \Theta$, combine all simple support functions with focal element $\{i\}$ into a single simple support function with the same focal element, $m^i(\cdot)$.

$$m^i = m_1^i \oplus m_2^i \oplus \cdots \tag{5.5}$$

When simple support functions with homogeneous focal elements are combined, Dempster's rule reduces to the following formulas.

$$m^i(i) = 1 - \prod_j \left[1 - m_j^i(i)\right] \tag{5.6}$$

$$m^i(\Theta) = \prod_j \left[1 - m^i_j(i)\right]$$

$$m^i(\cdot) = 0.0 \text{ for all other focal elements}$$

Similarly, for each $i \in \Theta$, combine all simple support functions with focal element $\neg\{i\}$ into a single simple support function with the same focal element, $m^{\neg i}(\cdot)$.

$$m^{\neg i} = m_1^{\neg i} \oplus m_2^{\neg i} \oplus \cdots \tag{5.7}$$

The above equations also can be applied to combine these simple support functions.

$$m^{\neg i}(\neg i) = 1 - \prod_j \left[1 - m_j^{\neg i}(\neg i)\right] \tag{5.8}$$

$$m^{\neg i}(\Theta) = \prod_j \left[1 - m_j^{\neg i}(\neg i)\right]$$

$$m^{\neg i}(\cdot) = 0.0 \text{ for all other focal elements}$$

At the end of this step, there are $2n$ simple support functions, half of which are focused on the singleton propositions and the other half are focused on the complements of the singletons.

2) After the simple support functions with homogeneous evidence are combined, each pair of functions that focus on a singleton and its complement are combined into a new belief function. For each $i \in \Theta$, let $M^i = m^i \oplus m^{\neg i}$. There are n of these new belief functions, one for each member of the FOD. Barnett calls these functions *simple evidence functions* (SEFs). Applying Dempster's rule to this special case, the following combination functions are easily derived.

$$M^i(i) = K m^i(i) m^{\neg i}(\Theta) \tag{5.9}$$

$$M^i(\neg i) = K m^i(\Theta) m^{\neg i}(\neg i)$$

$$M^i(\Theta) = K m^i(\Theta) m^{\neg i}(\Theta)$$

where $K = 1 - m^i(i)m^{\neg i}(\neg i)$. It is easy to see that the belief functions produced by these equations are special cases of the dichotomous belief functions defined earlier. Thus, the evidence in these belief functions are focused entirely on $\{i, \neg i, \Theta\}$, for $i \in \Theta$. As will be described later, the dichotomous nature of these belief functions is exploited in PSEIKI's label-based evidence accumulation scheme to efficiently combine a large number of belief functions. Because the

SEFs created in this step are valid belief functions, there is no requirement that they be created from the combination of simple support functions using equation (5.9). If belief functions with the same form as SEFs† need to be combined, then the equations given in step 3 can be used to generate the appropriate values.

3) In general, an exponential time algorithm is required to combine the *n* SEFs produced by the formulas in step 2. However, Barnett derived a number of equations to compute specific quantities associated with the belief function resulting from the combination of the above SEFs. That is, quantities associated with $m = M^1 \oplus \cdots \oplus M^n$ can be computed in linear time (such as the mass-of or belief-in subsets of Θ). For example, the equations Barnett derived to determine the probability mass associated with arbitrary subsets of Θ in linear time are shown below. Equation (5.10) provides the probability mass for singleton propositions.

$$m(i) = K\left[M^i(i)\prod_{j\neq i}(1 - M^j(j)) + M^i(\Theta)\prod_{j\neq i}M^j(\neg j)\right] \qquad \text{for } i \in \Theta \tag{5.10}$$

Equation (5.11) provides the probability mass for nonsingleton propositions.

$$m(\psi) = K\prod_{i\in\psi} M^i(\Theta)\prod_{i\notin\psi} M^i(\neg i) \qquad \text{for } \psi \subseteq \Theta, \text{ and } |\psi| \geq 2 \tag{5.11}$$

where

$$K^{-1} = \left[\prod_i (1 - M^i(i))\right]\left[1 + \sum_i \frac{M^i(i)}{1 - M^i(i)}\right] - \prod_i M^i(\neg i) \tag{5.12}$$

Of course, the null proposition receives zero mass.

$$m(\varnothing) = 0 \tag{5.13}$$

Note that these formulas require linear time to compute the probability mass associated with any particular subset of Θ. However, if the masses associated with all subsets of Θ are desired, the algorithm must be applied an exponential number of times, negating its efficiency (because there are $2^{|\Theta|}$ subsets). This limitation does not cause any difficulties in PSEIKI because the beliefs in only a few predetermined subsets are needed. The formula to compute the belief in a nonsingleton proposition in linear time is shown below (the belief in a singleton is equal to its

† i.e., dichotomous belief functions with the singleton propositions and their compliments as the focal elements.

probability mass).

$$\mathrm{Bel}(\psi) = K\left[\prod_i (1-M^i(i)) \sum_{i \in \psi} \frac{M^i(i)}{(1-M^i(i))} + \prod_{i \in \psi} M^i(\neg i) \prod_{i \in \psi} (1-M^i(i)) - \prod_i M^i(\neg i)\right] \tag{5.14}$$

Barnett also derived equations to determine the commonality numbers of a belief function; however, those equations are not needed here.

Because the desired quantities are computed as they are needed, the SEFs are never explicitly combined using equation (5.3). Instead, the SEFs are stored as such and combined by using the equations (5.10) through (5.14) whenever the values for accumulated belief are needed. Thus, the final belief function, $m(\cdot)$, can be represented as a collection of n SEFs $M^1, \cdots, M^n$. Furthermore, only 2N real numbers are needed to completely represent the final belief function because each SEF must have unity mass sum. Belief functions represented by a collection of SEFs in this manner will be called *composite belief functions*.

Barnett's three step linear time accumulation process is made possible by the invariance of Dempster's rule with respect to the order of combination. For example, if the following sequence of belief functions is combined in the order shown below, then the original version of Dempster's rule must be used to accumulate their evidence, resulting in exponential time complexity.

$$m(\cdot) = m^1_{12} \oplus m^6_3 \oplus m^4_9 \oplus m^8_{27} \oplus \cdots$$

However, by exploiting the order invariance of Dempster's rule, the three step process can be used to accumulate the functions' evidence. By grouping the belief functions in the following order, Barnett's equations become applicable.

$$m(\cdot) = \left[\left(m^1_1 \oplus m^1_2 \oplus \cdots\right) \oplus \left(m^{\neg 1}_1 \oplus m^{\neg 1}_2 \oplus \cdots\right)\right] \oplus \left[\left(m^2_1 \oplus m^2_2 \oplus \cdots\right) \oplus \left(m^{\neg 2}_1 \oplus m^{\neg 2}_2 \oplus \cdots\right)\right] \oplus \cdots$$

Step 1 of the process is used to condense the simple support functions with homogeneous focus.

$$m(\cdot) = \left[m^1 \oplus m^{\neg 1}\right] \oplus \left[m^2 \oplus m^{\neg 2}\right] \oplus \cdots$$

Step 2 of the process combines the $2n$ simple support functions into n dichotomous belief functions (simple evidence functions).

$$m(\cdot) = M^1 \oplus M^2 \oplus \cdots$$

Finally, step 3 of the process is used to determine the value of specific relevant quantities of the final belief function.

By exploiting the order invariance of Dempster's rule, it is possible to combine two belief functions in linear time if they were computed using Barnett's scheme and are represented as a collection of n SEFs. For example, assume that the following two belief functions, m_1 and m_2, have the same FOD and were derived from simple support functions focusing on singleton propositions and their complements.

$$m_1(\cdot) = M_1^1 \oplus M_1^2 \oplus \cdots \oplus M_1^n \tag{5.15}$$
$$m_2(\cdot) = M_2^1 \oplus M_2^2 \oplus \cdots \oplus M_2^n$$

In these equations, M_j^i is the SEF from the j^{th} belief function focusing on the i^{th} singleton proposition in the FOD. Combining these two belief functions results in the following formulas.

$$\begin{aligned} m(\cdot) &= m_1 \oplus m_2 \\ &= (M_1^1 \oplus M_1^2 \oplus \cdots \oplus M_1^n) \oplus (M_2^1 \oplus M_2^2 \oplus \cdots \oplus M_2^n) \end{aligned} \tag{5.16}$$

By exploiting the order invariance of Dempster's rule, it is possible to form pairs of SEFs with homogeneous focal elements.

$$m(\cdot) = (M_1^1 \oplus M_2^1) \oplus (M_1^2 \oplus M_2^2) \oplus \cdots \oplus (M_1^n \oplus M_2^n) \tag{5.17}$$

Each pair of SEFs can be combined using the special case of Dempster's rule shown in equation (5.4). Combining each pair of SEFs takes constant time and produces a new SEF with equivalent focal elements.

$$= M^1 \oplus M^2 \oplus \cdots \oplus M^n \tag{5.18}$$

Where $M^i = M_1^i \oplus M_2^i$ is the i^{th} SEF from the final belief function. Note that the above belief function is in the form used in step (2) of Barnett's scheme; thus, the equations of step (3) can be used to retrieve relevant quantities of the final belief function. Because each of the n SEF combinations can be performed in constant time, the total time to combine the original belief functions is O(n).

In summary, Barnett uses three mechanisms to permit the efficient combination of belief functions. First, he assumes that incoming evidence is limited to a small number of focal elements. Second, the associative and commutative nature of Dempster's rule permits the accumulation of subsets of the belief functions in an efficient manner. Third, he assumes that only a small number of the quantities of the final belief function are needed by the inference system. All three of these techniques are exploited in the accumulation process developed for PSEIKI.

5.2.2. Other Efficient Implementations of Dempster's Rule

Gordon and Shortliffe also were able to improve the computational complexity of the D-S theory by making an assumption about the type of evidence allowed to update beliefs [GorSho85]. They formed what they termed a hierarchical hypothesis space, a hierarchical partition of an element's FOD, and assumed that all evidence either would confirm or deny elements in the hierarchical partition. An example of a hierarchical hypothesis space that could be used in a computer vision system is shown in Fig. 5.1; it shows the partition that could be used by a target identification system to classify objects detected in a sequence of image frames. The identification system could use the partition shown in this figure if it detected an object that moved from frame to frame. If the system detected a moving object, then it would be able to use the hierarchy to provide evidence asserting that the object was a vehicle without needing to specify which type of vehicle. A system using the formulas derived by Barnett would not be able to provide evidence directly for the generic class of vehicles, because evidence is limited to focusing on the individual members of the FOD in that scheme. In order to provide a computational gain, Gordon and Shortliffe were forced to approximate Dempster's sum; the resulting approximation had a number of drawbacks. When presented with highly contradictory evidence, the approximation produced poor results. The approximation also prevented the computation of belief values for negations of elements in the hierarchy; thus, plausibilities for elements in the hierarchy could not be computed. Finally, the hierarchical decomposition used by the scheme required that the class hierarchy be strict. That is, the classes on the lower levels of the hierarchy could only have one parent.

Shafer and Logan were able to formalize the problem of using Dempster's rule to combine evidence focused on elements of a strict hierarchical partition of the FOD [ShaLog87]. By applying variations of Barnett's formulas to elements in a hierarchical

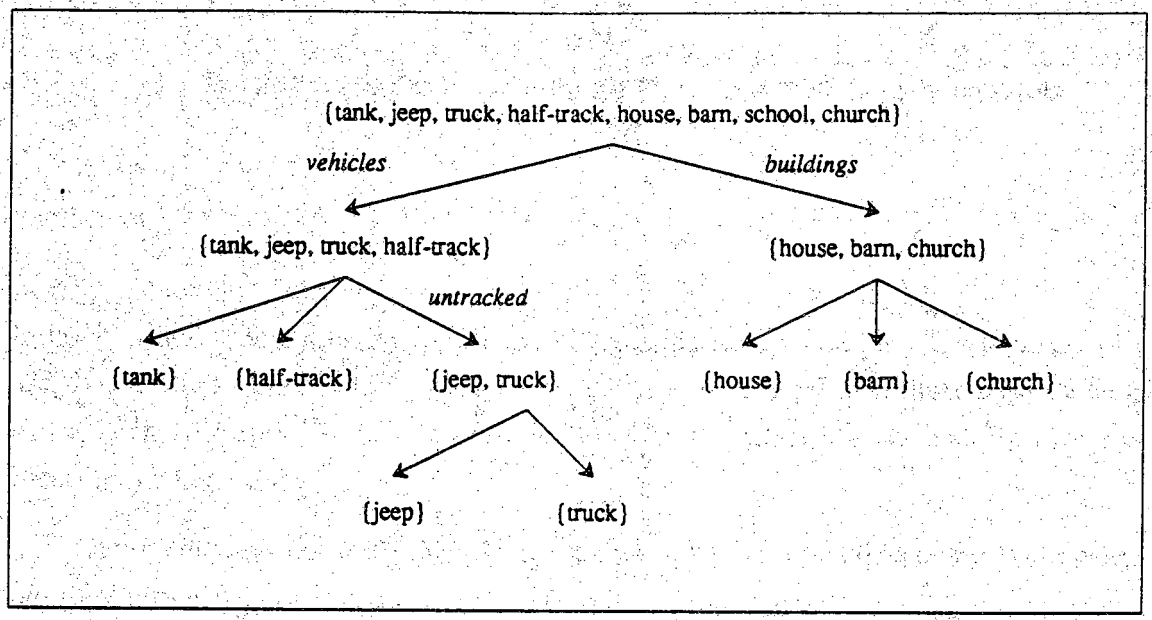


Figure 5.1 This figure shows a hierarchical partition of a hypothesis space that could be used to classify objects detected in an image of an outdoor scene.

partition of an element's FOD, they were able to compute Dempster's sum for elements in the partition without any approximations; thus, the results that their formulas provide are always valid. Their formulas also are slightly more general than those used by Gordon and Shortliffe in that they can compute both belief values and plausibilities for elements in the hierarchy.

Dichotomous belief functions involve the most drastic restriction to the D-S theory, but they provide the greatest computational gain. Obviously, since the belief functions can be treated as if they had binary FODs (i.e. $|\Theta| = 2$) [SafGot90], the time needed to combine two BPA's is constant. Thus the time needed to combine N BPA's is O(N).

Although the above variations of Dempster's rule greatly improve its computational efficiency, none of them is directly applicable to the problem of accumulating evidence in PSEIKI. Dichotomous belief functions are too restrictive to be used in a general matching procedure; their requirement that all probability mass be constrained to three subsets of Θ severly limits their applicability. Barnett's scheme, while remaining general enough for use in PSEIKI, is still too inefficient, if applied directly, to handle the

overwhelming amount of data in an image. Finally, the use of hierarchical hypothesis spaces is not possible because the hierarchy used in PSEIKI is not strict (e.g. an edge can be the child of two faces -- the faces it separates). For this reason, a new procedure to accumulate evidence was developed by incorporating the concept of an element's label into the reasoning process.

5.3. Two Label-Driven Models for Belief Accumulation

We will now describe two models for evidence accumulation, the first to introduce the reader to the notion of using labels for belief computation and the second to describe and exemplify the exact method used in PSEIKI. Both these models are motivated by the fact that Barnett's formulas can not be used for computer vision applications, despite their computational efficiencies, due to the massive amount of data involved.

MODEL 1:

In this model, we assume that practical considerations, such as computational feasibility, do not inhibit each source from providing evidence about all truth or falsity of all the propositions that can be discerned by the FOD. We further assume in this model that each source is structured in such a manner that it expresses its evidence as a combination of SEFs (as is assumed in Barnett's formalism). To explain, consider the following situation:

> You are sitting in a dimly lit bar and trying to determine the color of an object you are holding. You know *a-priori* the color can only be one of the following set of *labels*:
>
> $$\Theta = \{\text{green, black, blue, red}\}$$
>
> Despite their being in various states of inebriation, three of your friends, Bob, Jim, and Sue, are available to serve as experts to help you determine the correct label. Your mission is to elicit information from them and determine the correct label, meaning the most believed color of the object. You are not allowed to use your own judgements, except for having defined the FOD.

Due to nature of the algorithm you have programmed into your laptop computer, you insist that the experts help you by providing answers to the following dichotomous

questions about each possible label. In order words, you will pose the following questions to your friends:

RED1: What's your belief that the color of the object is red?

RED2: What's your belief that the color of the object is not red?

You also will ask equivalent questions about the other colors in the FOD. We will refer to each such pair of questions as a dichotomous set. You will insist that each expert treat each pair such questions in isolation from the other pairs and apportion all his/her belief between the possibilities represented by the pair of questions; of course, the expert is allowed to withhold some or all of his/her belief. Therefore, when, say, Bob is asked if the color is/isn't red, we have

$$M^{red}_{Bob}(red) + M^{red}_{Bob}(\neg red) + m^{red}_{Bob}(\Theta) = 1.0$$

Each pair of dichotomous questions elicits a SEF for each of the experts. All the SEFs provided by a single expert constitute a composite belief function as expressed by that expert.

Let's assume that the composite beliefs expressed by all the experts are as displayed in table 5.1. To combine all this evidence, we could use Barnett's formulas and benefit from its computational efficiencies. However, as we will show, it is possible to employ a computational scheme that uses the formulas of Barnett as a starting point and then through iterative updating yields the final belief function even more efficiently. This new computational scheme uses the notion of labels in the following manner. We combine the SEFs corresponding to the first expert, and call the most believed element of Θ the *current label*. From the other composite belief functions, we now accumulate only those SEFs that focus on this label into the overall belief function.† If the label has remained

† Lest the reader be alarmed by the apparent grossness of this approach vis-a-vis the direct application of Barnett's formulas, in which all the SEFs would be combined, in appendix A we have used Monte Carlo techniques to show that the label-based scheme produces erroneous results only slightly more often than the direct approach. The important point to note is that the computational infeasibility may preclude the implementation of the direct scheme; better to use an approach that is implementable even if there is a slight probability that it might produce incorrect results. As the reader will realize, if mistakes are made at one level of PSEIKI's hierarchy, they can be corrected by the constraints invoked at a higher level of the hierarchy. In a sense, one could say that in PSEIKI we have implemented a scheme that is likely to work correctly in a vast majority of cases, and when errors are made, there are mechanisms available for correction. Yet, with some small probability, whose value is unknown at this time, PSEIKI could produce an incorrect interpretation. However, that is the price that must be paid for computational feasibility. Note that this behavior of PSEIKI is not unlike human cognition. In human reasoning at any level of detail, we are capable of making erroneous judgments, which in may cases get rectified by invoking higher level considerations.

Table 5.1 This table shows the evidence provided by Bob, Jim and Sue about the object's color.

Friend	Evidence		Equation
Bob	$M^{green}_{Bob}(green) = 0.05$	$M^{black}_{Bob}(black) = 0.10$	(5.19)
	$M^{green}_{Bob}(\neg green) = 0.75$	$M^{black}_{Bob}(\neg black) = 0.65$	
	$M^{green}_{Bob}(\Theta) = 0.20$	$M^{black}_{Bob}(\Theta) = 0.25$	
	$M^{blue}_{Bob}(blue) = 0.10$	$M^{red}_{Bob}(red) = 0.35$	
	$M^{blue}_{Bob}(\neg blue) = 0.72$	$M^{red}_{Bob}(\neg red) = 0.25$	
	$M^{blue}_{Bob}(\Theta) = 0.18$	$M^{red}_{Bob}(\Theta) = 0.40$	
Jim	$M^{green}_{Jim}(green) = 0.15$	$M^{black}_{Jim}(black) = 0.18$	(5.20)
	$M^{green}_{Jim}(\neg green) = 0.60$	$M^{black}_{Jim}(\neg black) = 0.50$	
	$M^{green}_{Jim}(\Theta) = 0.25$	$M^{black}_{Jim}(\Theta) = 0.32$	
	$M^{blue}_{Jim}(blue) = 0.11$	$M^{red}_{Jim}(red) = 0.24$	
	$M^{blue}_{Jim}(\neg blue) = 0.60$	$M^{red}_{Jim}(\neg red) = 0.18$	
	$M^{blue}_{Jim}(\Theta) = 0.29$	$M^{red}_{Jim}(\Theta) = 0.58$	
Sue	$M^{green}_{Sue}(green) = 0.02$	$M^{black}_{Sue}(black) = 0.29$	(5.21)
	$M^{green}_{Sue}(\neg green) = 0.88$	$M^{black}_{Sue}(\neg black) = 0.24$	
	$M^{green}_{Sue}(\Theta) = 0.10$	$M^{black}_{Sue}(\Theta) = 0.47$	
	$M^{blue}_{Sue}(blue) = 0.15$	$M^{red}_{Sue}(red) = 0.24$	
	$M^{blue}_{Sue}(\neg blue) = 0.56$	$M^{red}_{Sue}(\neg red) = 0.26$	
	$M^{blue}_{Sue}(\Theta) = 0.29$	$M^{red}_{Sue}(\Theta) = 0.50$	

unchanged after going through all the experts in this manner, then the label becomes the solution to the problem. For example, assume that the label 'red' was chosen for the color of the object, using the SEFs from equation (5.19) that were provided by Bob. If Jim and Sue's SEFs focusing on red, $M^{red}()$, were accumulated into the belief function and the label did not change, you would declare that the color of the object is red and would associate a belief value with this conclusion. If, on the other hand, during this process of combining Jim and Sue's evidence confirming or denying the label red, maximal belief was accorded to another color, such as blue, then the current label of the object would change from red to blue. Subsequently, the evidence would be combined by using only those SEFs from the remaining experts that either confirm or deny blue.

As we show in appendix A, when $|\Theta| = n$, the complexity associated with using Barnett's formula directly is $O(n \times N)$, while the complexity associated with the label based approach is $O(n + N)$, where N is the number of composite belief functions combined. For computer vision applications, this reduction in complexity makes it possible to carry out the experiments that would not be otherwise possible.

MODEL 2:

Our main goal in discussing Model 1 was merely to introduce the reader to the notion of accumulating beliefs through labels. Of course, all the belief functions in that model could be combined by a straightforward application of Barnett's formulas; however, the reader would probably want to use our label-based scheme for computer vision applications even though it is not quite as robust.

Now that we have introduced the reader to the notion of using labels for evidence accumulation, we are ready to present a second model, which is more structured than the first and which appears to lend itself naturally to a label based scheme. That is, it is necessary to use labels in model 2's accumulation process due to the manner in which evidence is accumulated. We will introduce the model with the help of the following experiment:

> Let's say that of the three friends, Bob appears to be the least tipsy and therefore is probably the best source of detailed evidence about the color of the object, detailed in the sense that Bob is capable of answering all the questions put to him in the dichotomous form shown above. On the other hand, the other two friends are too inebriated and have too short an attention span to answer more than a

couple of questions at a time. Given this scenario, you structure the evidence accumulation process in the following manner: You decide to combine all the SEF's supplied by Bob and use the most believed of the elements from Θ as the current label regarding the color of the object. Now the questions you ask of the other two friends are limited to eliciting their beliefs in either the confirmation or the denial of the current label. The two SEFs from Jim and Sue that posit beliefs in the three possibilities {label, $\neg$label, Θ} are then combined with the SEFs corresponding to all the evidence supplied by Bob. If the current label does not change through this update, you will accept that label as the final label.

The important point of difference between the two models follows: In the former model, all the experts supply confirmation/denial evidence for all the possible labels; these may then be combined by either a direct application of Barnett's formulas or by using our label based procedure. On the other hand, in the latter model, a distinction is made between the experts. On grounds of practical considerations, one of the experts is treated as a provider of initial evidence regarding the credibility of all the possibilities in the FOD -- although in order to facilitate the use of computationally efficient procedures this expert must provide evidence in answer to dichotomous queries regarding each of the possibilities. The rest of the experts are then treated as updaters of the most believed conclusion drawn from the first expert, this updating may either confirm or disconfirm the conclusion.

We believe that the approach to evidence accumulation exhibited by model 2 naturally describes the flow of evidence in verification systems. Initially, some sensor or expert provides evidence about the different possibilities and the system forms an initial hypothesis based on that evidence. For example, a satellite sensor might tell us that an object detected on the ground is either a nuclear tipped mobile rocket, a tanker truck, etc., with varying degrees of belief. Again, in order to use computationally efficient procedures, this inflow of initial evidence may be structured in such a manner that it is composed of a collections of SEFs, each confirming or denying each possibility in the frame of discernment. (Such structuring occures naturally in systems that use sets of templates, numerical, symbolic or conceptual, for each possibility. The application of each template would then yield an SEF for that possibility and the collection of all the SEFs would constitute a composite belief function for that sensor or expert.) After the initial hypothesis has been formed, other sensors/experts are queried to verify the validity of the initial hypothesis. In our example, suppose the most believed hypothesis on the basis of the

satellite report is that the object is a nuclear tipped mobile rocket, we might then decide to drop a sensitive radiation detector in the vicinity of the rocket and use its evidence to either confirm or deny the hypothesis. Note that it may not be feasible to generate the verification evidence until the initial hypothesis has been formed. In the example, it may not be feasible to saturate an area believed to contain missiles with the radiation detectors; they can only be dropped on objects already believed to be missiles. The important point being made here that there will be situations that do not allow for all sources of evidence to be treated in a uniform manner -- some must be used for initialization, while others are used for confirming or refuting the hypotheses generated by those that were used for initialization.

The method of evidence accumulation used in PSEIKI corresponds to model 2. Initially, on the basis of only spatial proximity considerations, image element to model element comparisons are used for generating initial hypotheses about the various possible model labels for the image elements. Subsequently, consistency considerations between image elements are used for either confirming or refuting these hypotheses. We must hasten to add that while theoretically we could set up the PSEIKI evidence accumulation process to correspond to Model 1 and use Barnett's formulas to compute the final belief functions without error, that is not computationally feasible. In other words, the updating process could be made to yield SEFs that span the entire FOD, however the resulting computation would be too burdensome and would make PSEIKI unusable.

5.4. A Detailed Description of Label Based Schemes for Evidence Accumulation

The previous section described the motivating factors for the development of PSEIKI's evidence accumulation process; this section will describe the process in detail. As has already been mentioned, the first motivating factor for the accumulation process' development was the overwhelming amount of data present in most images. Although Barnett's scheme with linear complexity (in the size of the FOD) yields a great improvement over the exponmential complexity of the original formulation of Dempster's rule, his formulas are still too inefficient when a large number of belief functions with large FODs are being combined. The label-based scheme further reduces the computational complexity of Dempster's rule by splitting the accumulation process into two phases: initialization and updating. During the initialization phase, a belief function is built using Barnett's formalism by accumulating confirmatory and disconfirmatory evidence focused

on all members of the FOD. Once this initial belief function has been established, the updating phase commences and the focus of all new evidence is restricted to the current label. The second motivation for the development of label-based accumulation procedure stems from the hypothesize/verify nature of PSEIKI's evidence accumulation process. As mentioned in the discussion of model 2, this form naturally lends itself to the use of labels in the reasoning process.

Assume that the goal of the accumulation process is to determine the identity of an element[†], E, from the set of *n* possibilities, $\theta_1, \theta_2, \cdots, \theta_n$. Therefore, the FOD for E is

$$\Theta = \{\theta_1, \theta_2, \cdots, \theta_n\}$$

Furthermore, assume that N composite belief functions, $Bel_1(\cdot), Bel_2(\cdot), \cdots, Bel_N(\cdot)$, have been produced, each a combination of n simple evidence functions focusing on the singleton propositions and their compliments. In other words, each belief function is assumed to be represented as a collection of *n* SEFs containing information about the element's identity. Using the notation from section 5.2.1, the BPA for j^{th} belief function, $Bel_j(\cdot)$, can be expressed as

$$m_j(\cdot) = M_j^1 \oplus M_j^2 \oplus \cdots \oplus M_j^n \tag{5.22}$$

Using the example of the previous section, assume that you and your friends, Bob, Jim and Sue, are trying to determine the color of an object. Remember, the element can be one of four possible colors, $\Theta = \{\text{green, black, blue, red}\}$. Assume that your friends can answer the dichotomous questions about the color, and thus can provide independent composite belief functions, Bel_{Bob}, Bel_{Jim} and Bel_{Sue}, containing information about the element's identity; each of these composite belief functions are represented as four SEFs. The goal of the accumulation process is to combine the information contained in these three composite belief functions into a single belief function and form a hypothesis about the element's color based on this final belief function. Assume that the belief functions are composed of the SEFs shown in table 5.1.

During the initialization phase of the accumulation process, one composite belief function distinguished from the rest; this belief function is called the element's initial belief function. The remaining N-1 composite belief functions are used in the second phase of the accumulation procedure and are called the updating belief functions.

[†] We refer to E as an element because in PSEIKI, we are interested in determining model labels for image *elements*.

Without loss of generality, assume that the first composite belief function, $Bel_1(\cdot)$, is used as the initial belief function, and that the rest are used as updating belief functions. After the initial belief function for the element's identity is computed, the element's *label* is determined. An element's label is defined to be the singleton proposition with the largest belief. (If two or more elements of the FOD yield the same belief, one is selected arbitrarily.) Thus, in some sense, the element's label can be considered to be the current best hypothesis for the element's identity. As an example of the ease with which an element's label can be found, consider the process of determining the label for element E. Remember that the FOD for E consists of *n* elements

$$\Theta = \{\theta_1, \theta_2, \cdots, \theta_n\}$$

To determine E's label, the element of Θ with maximum belief must be found. However, since only singletons are being considered as labels, finding the member of Θ with greatest belief is equivalent to finding the member of Θ with the largest probability mass (the belief of a singleton is equal to its probability mass). Thus, only the following elements of E's BPA need be considered

$$m_E(\theta_\alpha) \quad \text{for} \quad \alpha = 1, \ldots, n$$

Let $\theta_{\alpha_{max}}$ be the singleton proposition with greatest belief. That is,

$$m_E(\theta_{\alpha_{max}}) \geq m_E(\theta_\alpha) \quad \text{for} \quad \alpha = 1, \ldots, n \tag{5.23}$$

The label of element E is defined to be $label = \theta_{\alpha_{max}}$ †. In the example, if $Bel_{Bob}(\cdot)$ was chosen as the initial belief function, equation (5.10) could be used to determine the belief in each of the singleton propositions.

$$\begin{aligned} Bel_{initial}(\text{green}) &= 0.0607 \\ Bel_{initial}(\text{black}) &= 0.1090 \\ Bel_{initial}(\text{blue}) &= 0.0946 \\ Bel_{initial}(\text{red}) &= 0.5001 \end{aligned} \tag{5.24}$$

Thus, the initial label for the element's color would be 'red' with belief 0.5.

Once the initialization phase of the accumulation scheme is complete and E's initial belief function has been computed, the belief updating phase begins. In this phase,

† A process for determining the probability mass of all the singletons (and hence the element's label) in linear time will be described in Section 5.4.1.

evidence from the remaining N-1 composite belief functions is accumulated into the element's belief function. However, the evidence contained in these belief functions is not blindly combined with the initial belief function, as would be done in Barnett's scheme. Equation (5.25) shows how the composite belief functions would be combined by using a direct application of Barnett's scheme.

$$\begin{aligned} m_{final} = &\left\{ M_1^1 \oplus M_1^2 \oplus \cdots \oplus M_1^{label} \oplus \cdots \oplus M_1^n \right\} \oplus \\ &\left\{ M_2^1 \oplus M_2^2 \oplus \cdots \oplus M_2^{label} \oplus \cdots \oplus M_2^n \right\} \oplus \\ &\vdots \\ &\left\{ M_N^1 \oplus M_N^2 \oplus \cdots \oplus M_N^{label} \oplus \cdots \oplus M_N^n \right\} \end{aligned} \tag{5.25}$$

Each of the rows for equation (5.25) represents one of the composite belief functions being combined (remember, M_j^i is the SEF focusing on the i^{th} singleton from the j^{th} belief function). In particular, the top row is the initial belief function and the rest of the rows are the updating belief functions. Regrouping the SEFs from the updating belief functions allows them to be combined using equation (5.4), yielding

$$\begin{aligned} m_{final} = &\left\{ M_1^1 \oplus M_1^2 \oplus \cdots \oplus M_1^{label} \oplus \cdots \oplus M_1^n \right\} \oplus \\ &\left\{ M_2^1 \oplus M_3^1 \oplus \cdots \oplus M_N^1 \right\} \oplus \\ &\vdots \\ &\left\{ M_2^{label} \oplus M_3^{label} \oplus \cdots \oplus M_N^{label} \right\} \oplus \\ &\vdots \\ &\left\{ M_2^n \oplus M_3^n \oplus \cdots \oplus M_N^n \right\} \end{aligned} \tag{5.26a}$$

Once again, the first row represents the initial belief function; however, the remaining rows (those containing the updating belief functions) have been rearranged such that all of the SEFs with the same focal elements are now in the same group. Equation (5.4) can now be used to collapse each group of SEFs from the updating belief functions into a

single updating SEF.

$$m_{final} = \left\{ M_1^1 \oplus M_1^2 \oplus \cdots \oplus M_1^n \right\} \oplus M_{update}^2 \oplus \cdots \oplus M_{update}^{label} \oplus \cdots \oplus M_{update}^n \qquad (5.26b)$$

where $M_{update}^i = M_2^i \oplus M_3^i \oplus \cdots \oplus M_N^i$, the updating SEF focusing on the i^{th} singleton, is computed by applying equation (5.4) N-2 times. The SEFs from the updating belief functions focusing on the label element, M_{update}^{label}, are sometimes denoted as m_{update}.

When incorporating updating evidence, the computational burden can be eased by splitting the updating phase into a number of *updating cycles*. At the start of each updating cycle, the element's label is computed using the previously described procedure. After the label has been computed, only the updating SEFs focusing on the label element are accumulated into the overall belief function. That is, from each updating belief function, only the SEF with focus elements $\{\theta_{label}\}$, $\{\neg\theta_{label}\}$ and $\{\Theta\}$ is accumulated in the element's belief function. Thus, if a singleton never becomes the element's label over the course of the updating cycles, then the updating SEFs focusing on it will not be accumulated into the element's belief function†. The rationale behind accumulating only the SEF focusing on the label element follows. If the assigned label is correct, then the confirmatory evidence contained in the SEFs accumulated into the element's belief function should reinforce the belief in the label, allowing the label to be left unchanged. If the label assigned to an element is incorrect, then the disconfirmatory evidence contained in the SEF accumulated into the element's belief function should be sufficient to force the label to change. Of course, if the label is incorrect, it would be advantageous to accumulate the SEFs focusing on the correct label; however, the correct label cannot be known *a-priori* (remember, the whole task of the system is to find the correct label). Thus, all new evidence is focused on either trying to prove or trying to disprove that an element's label is correct (i.e. that the element's identity has been correctly determined). In model 1, the limiting of information to focus on the current label element is akin to the use of nonadmissible heuristics in graph search procedures [Nil80]; in both cases, the accuracy of the final solution is sacrificed for a gain in computational efficiency.

During the first updating cycle, only the updating SEF from equation (5.26) focusing on the initial label is accumulated into the overall belief function.

---

† The consequences of accumulating only some of the updating SEFs into the element's belief function will be investigated in appendix A via a Monte Carlo simulation.

$$m_{overall} = \left\{ M_1^1 \oplus M_1^2 \oplus \cdots \oplus M_1^{label} \oplus \cdots \oplus M_1^n \right\} \oplus M_{update}^{label} \tag{5.27}$$

The updating SEF is accumulated into the overall belief function by grouping it with the SEF from the initial belief function also focusing on the label element.

$$m_{overall} = \left\{ M_1^1 \oplus M_1^2 \oplus \cdots \oplus (M_1^{label} \oplus M_{update}^{label}) \oplus \cdots \oplus M_1^n \right\} \tag{5.28}$$

The two SEFs are then combined using equation (5.4).

$$m_{overall} = \left\{ M_1^1 \oplus M_1^2 \oplus \cdots \oplus M_{final}^{label} \oplus \cdots \oplus M_1^n \right\} \tag{5.29}$$

The first updating cycle ends after the the updating SEF focused on the initial label is accumulated into the overall belief function. After the first updating cycle finishes, the second updating cycle starts and the belief function's label is determined again. If the updating evidence did not cause the belief in the label element to fall below the belief in another singleton, then the accumulation process is finished. In the previous example, updating evidence focused on the label element, 'red', from Bel_{Jim} and Bel_{Sue} would be combined using equation (5.4).

$$M_{update}^{red} = M_{Jim}^{red} \oplus M_{Sue}^{red}$$

The updating SEF's masses produced by this equation are:

$$M_{update}^{red}(red) = 0.3542$$
$$M_{update}^{red}(\neg red) = 0.3216$$
$$M_{update}^{red}(\Theta) = 0.3242$$

When this updating SEF is combined with the SEF from the initial belief function focusing on 'red', the following SEFs result:

$$\begin{aligned} & M^{green}(green) = 0.05 \quad M^{black}(black) = 0.10 \\ & M^{green}(\neg green) = 0.75 \quad M^{black}(\neg black) = 0.65 \\ & M^{green}(\Theta) = 0.20 \quad M^{black}(\Theta) = 0.25 \end{aligned}$$

$Bel_{overall}$: (5.30)

$$\begin{aligned} & M^{blue}(blue) = 0.10 \quad M_{final}^{red}(red) = 0.4746 \\ & M^{blue}(\neg blue) = 0.72 \quad M_{final}^{red}(\neg red) = 0.3631 \\ & M^{blue}(\Theta) = 0.18 \quad M_{final}^{red}(\Theta) = 0.1623 \end{aligned}$$

These SEFs produce the following singleton belief values:

$$\text{Bel(green)} = 0.0734$$

$$\text{Bel(black)} = 0.1247$$

$$\text{Bel(blue)} = 0.1019$$

$$\text{Bel(red)} = 0.5605$$

Thus, the element's final label would be 'red' with belief 0.56.

If the accumulated disconfirmatory evidence about an element's label is enough to force the belief in the label element to fall below the belief in another member of Θ, then the label will be changed to the element with greater belief. The updating SEFs focusing on the new label element and its complement are then accumulated into the belief function. For example, let label(1) denote the label given the element during the first updating cycle. Also, let label(2) denote the label given the element at the start of the second updating cycle. Thus if label(1) $\neq$ label(2), then the updating SEFs focusing on label(2) are accumulated into the belief function.

$$m_{\text{final}} = \left\{ M_1^1 \oplus M_1^2 \oplus \cdots \oplus M_1^n \right\} \oplus M_{\text{update}}^{\text{label(1)}} \oplus M_{\text{update}}^{\text{label(2)}} \tag{5.31}$$

This iterative procedure continues until the label at step *i* of the process, label(i), is equal to a previous label, label(j) for some $j \leq i$. The accumulation of updating evidence stops when this termination condition is met or when the updating SEFs focusing on all *n* singletons have been accumulated into the final belief function.

For example, if the belief function shown in equation (5.20) was modified slightly such that its SEF focusing on the label 'red' denied that red was the true color, then the element's label would change after the first updating cycle. Assume that Bel_{Jim} was modified such that it had the probability masses shown below in equation (5.32) (only the masses from the SEF focusing on 'red' have been changed); note that the modified SEF now denies red's validity. The other belief functions are assumed to be unchanged and have the masses shown in equations (5.19) and (5.21).

$$
\begin{array}{lll}
 & M^{green}_{Jim}(green) = 0.15 & M^{black}_{Jim}(black) = 0.18 \\
 & M^{green}_{Jim}(\neg green) = 0.60 & M^{black}_{Jim}(\neg black) = 0.50 \\
 & M^{green}_{Jim}(\Theta) = 0.25 & M^{black}_{Jim}(\Theta) = 0.32 \\
Bel_{Jim}': & & \\
 & M^{blue}_{Jim}(blue) = 0.11 & M^{red}_{Jim}(red) = 0.09 \\
 & M^{blue}_{Jim}(\neg blue) = 0.60 & M^{red}_{Jim}(\neg red) = 0.75 \\
 & M^{blue}_{Jim}(\Theta) = 0.29 & M^{red}_{Jim}(\Theta) = 0.16
\end{array}
\tag{5.32}
$$

With this new belief function, the updating SEF produced by combining the SEFs from Bel_{Jim} and Bel_{Sue} focusing on the color red would be:

$$M^{red}_{update} = M^{red}_{Jim} \oplus M^{red}_{Sue}$$

The updating SEF's masses produced by this equation are:

$$
\begin{aligned}
M^{red}_{update}(red) &= 0.1318 \\
M^{red}_{update}(\neg red) &= 0.7678 \\
M^{red}_{update}(\Theta) &= 0.1004
\end{aligned}
$$

When this updating SEF is combined with the SEF from the initial belief function focusing on red, the following SEFs result:

$$
\begin{array}{lll}
 & M^{green}(green) = 0.05 & M^{black}(black) = 0.10 \\
 & M^{green}(\neg green) = 0.75 & M^{black}(\neg black) = 0.65 \\
 & M^{green}(\Theta) = 0.20 & M^{black}(\Theta) = 0.25 \\
Bel_{overall}: & & \\
 & M^{blue}(blue) = 0.10 & M^{red}_{final}(red) = 0.1919 \\
 & M^{blue}(\neg blue) = 0.72 & M^{red}_{final}(\neg red) = 0.7506 \\
 & M^{blue}(\Theta) = 0.18 & M^{red}_{final}(\Theta) = 0.0575
\end{array}
\tag{5.33}
$$

The procedure to be described in section 5.4.1 can now be used to determine the belief in the singletons.

$$Bel(green) = 0.1521$$

$$Bel(black) = 0.2518$$

$$Bel(blue) = 0.1994$$

$$Bel(red) = 0.2479$$

Thus, the element's label would change from red to black (with belief 0.25). A new updating cycle would then start to update the belief in the color black. The updating SEF produced by combining the SEFs from Bel_{Jim} and Bel_{Sue} focusing on black would be:

$$M_{update}^{black} = M_{Jim}^{black} \oplus M_{Sue}^{black}$$

The updating SEF's masses produced by this equation are:

$$M_{update}^{black}(black) = 0.2828$$

$$M_{update}^{black}(\neg black) = 0.5319$$

$$M_{update}^{black}(\Theta) = 0.1853$$

The following SEFs result from combining this updating SEF with the SEF from the initial belief function focusing on the color black.

$Bel_{overall}$:

		(5.34)
$M^{green}(green) = 0.05$	$M_{final}^{black}(black) = 0.1540$	
$M^{green}(\neg green) = 0.75$	$M_{final}^{black}(\neg black) = 0.7853$	
$M^{green}(\Theta) = 0.20$	$M_{final}^{black}(\Theta) = 0.0607$	
$M^{blue}(blue) = 0.10$	$M_{final}^{red}(red) = 0.1919$	
$M^{blue}(\neg blue) = 0.72$	$M_{final}^{red}(\neg red) = 0.7506$	
$M^{blue}(\Theta) = 0.18$	$M_{final}^{red}(\Theta) = 0.0575$	

The procedure to be described in section 5.4.1 can now be used to determine the belief in the singletons.

$$Bel(green) = 0.1905$$

$$Bel(black) = 0.2158$$

$$Bel(blue) = 0.2380$$

$$Bel(red) = 0.2688$$

In this iteration, the label is switched back to red (with belief 0.27). Because the updating evidence focusing on the color red already has been accumulated into the overall

belief function, the accumulation process stops. Thus, the final label for the element is red with belief 0.27.

As has been mentioned, the updating process can be viewed as being composed of a number of updating cycles. At the start of each cycle, the element's label is determined from the overall belief function. If the new label equals a previous label, then the updating process is over. However, if the label is assigned a singleton that it has not previously assumed, then the SEFs from the updating belief functions focusing on the label element are accumulated into the overall belief function. This iterative process continues until the previously described termination condition is met or all of the singletons have been used as labels. The accumulation process is expressed algorithmically below. The initial belief function is determined in step 1 of the algorithm (i.e. $i = 1$). Steps 2-5 comprise the belief updating loop; each iteration of the loop corresponds to one updating cycle.

1) Determine the initial belief function and let $i = 1$.

2) Determine the label for step i, label(i).

3) STOP if the updating SEFs focusing on the label already have been accumulated into the belief function. (i.e. label(i) = label(j) for some $1 \leq j < i$). Otherwise, using equation (5.4), accumulate the updating SEFs focusing on the label into the overall belief function.

4) Let $i = i + 1$

5) Go to step 2.

Note that this new accumulation process degenerates into Barnett's scheme if the element's label changes such that it assumes all possible labels in the element's FOD. However, if the number of labels assigned to an element is significantly smaller than the size of the FOD, then the label-based scheme will yield a notable increase in efficiency compared to the straightforward implementation of Barnett's formulas. In systems using model 2, if computing the SEFs from the updating belief functions is costly, a large computational gain can be achieved by postponing the generation of the updating belief functions' SEFs until they are needed. If the generation of these SEFs is postponed until they need to be accumulated into the overall belief function, then only those focusing on the current label will be generated during any updating cycle. Thus, only a few of the SEFs

will be generated because most of them never are accumulated into the final belief function. If computing these SEFs is costly, then a significant computational savings is achieved for this reason alone.

5.4.1. Computing Labels in Linear Time

Using the equations supplied by Barnett, the process for determining an element's label potentially requires $O(n^2)$ operations because an O(n) algorithm is used to determine the probability masses for each of the n singletons. Remember, the following formula from equation (5.10) yields the probability mass for a singleton proposition.

$$m(\theta_i) = K\left[M^i(\theta_i)\prod_{j\neq i}(1 - M^j(\theta_j)) + M^i(\Theta)\prod_{j\neq i}M^j(\neg\theta_j)\right]$$

where K does not depend on θ_i.

$$K^{-1} = \left[\prod_i(1 - M^i(\theta_i))\right]\left[1 + \sum_i \frac{M^i(i)}{1 - M^i(i)}\right] - \prod_i M^i(\neg i)$$

However, it is possible to determine the belief in all n singleton propositions in O(n) time. This new efficiency is achieved by precomputing some of the values used by equation (5.10) and reformulating the equation to compute the mass of a singleton using a constant number of operations. Thus, once the precomputed values are found, the belief in all n singletons can be computed in O(n) time. Furthermore, because the precomputed values can be found in O(n) time, a total of O(n) operations are needed to find an element's label. The values needed by the new formulation of Barnett's equation are called *partial mass products* and are defined below.

$$\Pi^+_>[\theta_i] = \prod_{j>i}(1 - M^j(\theta_j)) \quad \text{for } 1 \le i < n \tag{5.35}$$

$$\Pi^+_<[\theta_i] = \prod_{j<i}(1 - M^j(\theta_j)) \quad \text{for } 1 < i \le n \tag{5.36}$$

$$\Pi^-_>[\theta_i] = \prod_{j>i}M^j(\neg\theta_j) \quad \text{for } 1 \le i < n \tag{5.37}$$

$$\Pi^-_<[\theta_i] = \prod_{j<i}M^j(\neg\theta_j) \quad \text{for } 1 < i \le n \tag{5.38}$$

It is possible to determine the partial mass products for all n members of the FOD in O(n) time. For example, the n values of $\Pi^+_<[\cdot]$ can be found in O(n) time using the following

recursive definition.

$$\Pi^{+}_{\leq}[\theta_1] = 1 \tag{5.39}$$

$$\Pi^{+}_{\leq}[\theta_i] = (1 - M^{i-1}(\theta_{i-1}))\Pi^{+}_{\leq}[\theta_{i-1}] \quad \text{for } i > 1 \tag{5.40}$$

The two products used in Barnett's formula can be expressed in terms of partial mass functions.

$$\prod_{j \neq i}(1 - M^j(\theta_j)) = \Pi^{+}_{>}[\theta_i] \times \Pi^{+}_{\leq}[\theta_i] \tag{5.41}$$

$$\prod_{j \neq i} M^j(\neg\theta_j) = \Pi^{-}_{>}[\theta_i] \times \Pi^{-}_{\leq}[\theta_i] \tag{5.42}$$

Thus, once the partial mass functions and the conflict, K, have been determined, the probability mass of a singleton can be found using a constant number of operations.

$$m(\theta_i) = K\left[M^i(\theta_i)\Pi^{+}_{>}(\theta_i)\Pi^{+}_{\leq}(\theta_i) + M^i(\Theta)\Pi^{-}_{>}(\theta_i)\Pi^{-}_{\leq}(\theta_i)\right] \tag{5.43}$$

Note that K must be computed only once for all $1 \leq i \leq n$ because it does not depend on θ_i. Once the masses for all the singletons are found, the singleton with greatest mass is chosen as the label.

A simple example will be given to demonstrate the process of determining a label. This example will show the procedure used to find the label of the initial belief function from the last section. Recall that the first composite belief function, Bel_{Bob}, was used as the initial belief function in that example (the belief function is shown in equation (5.19)). The probability masses for its SEFs are shown below.

$$M_1^{green}(green) = 0.05 \qquad M_1^{black}(black) = 0.10$$
$$M_1^{green}(\neg green) = 0.75 \qquad M_1^{black}(\neg black) = 0.65$$
$$M_1^{green}(\Theta) = 0.20 \qquad M_1^{black}(\Theta) = 0.25$$

Bel_1:

$$M_1^{blue}(blue) = 0.10 \qquad M_1^{red}(red) = 0.35$$
$$M_1^{blue}(\neg blue) = 0.72 \qquad M_1^{red}(\neg red) = 0.25$$
$$M_1^{blue}(\Theta) = 0.18 \qquad M_1^{red}(\Theta) = 0.40$$

The partial mass products for these SEFs are:

$$\begin{aligned}
&\Pi^{+}_{\leq}[green] = 1.0 && \Pi^{+}_{>}[green] = 0.585\times0.9 = 0.5265 \\
&\Pi^{+}_{\leq}[black] = 0.95 && \Pi^{+}_{>}[black] = 0.65\times0.9 = 0.585 \\
&\Pi^{+}_{\leq}[blue] = 0.95\times0.9 = 0.855 && \Pi^{+}_{>}[blue] = 0.65 \\
&\Pi^{+}_{\leq}[red] = 0.855\times0.9 = 0.7695 && \Pi^{+}_{>}[red] = 1.0
\end{aligned} \tag{5.44}$$

$\Pi_{\prec}[\text{green}] = 1.0$ $\quad\quad$ $\Pi_{\succ}[\text{green}] = 0.18\times65 = 0.117$

$\Pi_{\prec}[\text{black}] = 0.75$ $\quad\quad$ $\Pi_{\succ}[\text{black}] = 0.25\times0.72 = 0.18$

$\Pi_{\prec}[\text{blue}] = 0.75\times0.65 = 0.4875$ $\quad\quad$ $\Pi_{\succ}[\text{blue}] = 0.25$

$\Pi_{\prec}[\text{red}] = 0.4875\times0.72 = 0.351$ $\quad\quad$ $\Pi_{\succ}[\text{red}] = 1.0$

The renormalization factor can be computed using equation (5.12).

$$K^{-1} = (0.95)(0.9)(0.9)(0.65)\left[1+\frac{0.05}{0.95}+\frac{0.1}{0.9}+\frac{0.1}{0.9}+\frac{0.35}{0.65}\right] - (0.75)(0.65)(0.72)(0.25)$$

$$= 0.5001 \times 1.8133 - 0.0878 \qquad (5.45)$$

$$= 0.8192$$

After all of the precomputed values have been found, the belief in each singleton can be found in constant time using equation (5.43).

$$\text{Bel(green)} = \left[(0.05)(0.5265)(1.0) + (0.20)(0.117)(1.0)\right] / 0.8192 \qquad (5.46)$$

$$= 0.0607$$

$$\text{Bel(black)} = \left[(0.10)(0.585)(0.95) + (0.25)(0.18)(0.75)\right] / 0.8192$$

$$= 0.1090$$

$$\text{Bel(blue)} = \left[(0.10)(0.65)(0.855) + (0.18)(0.25)(0.4875)\right] / 0.8192$$

$$= 0.0946$$

$$\text{Bel(red)} = \left[(0.35)(1.0)(0.7695) + (0.40)(1.0)(0.351)\right] / 0.8192$$

$$= 0.5001$$

Thus, the label for this belief function would be 'red' with belief 0.5.

5.5. Hierarchical Evidence Accumulation in PSEIKI

If the task of a system is to determine the identity of a number of elements which are arranged in a *part-of* hierarchy, then the evidence accumulation scheme introduced in the previous section can be embedded into the hierarchy to provide further computational gain. A part-of hierarchy is shown in panel (b) of Fig. 5.2; in this figure, as in most part-of hierarchies, elements on the higher levels of the hierarchy are defined by groups of

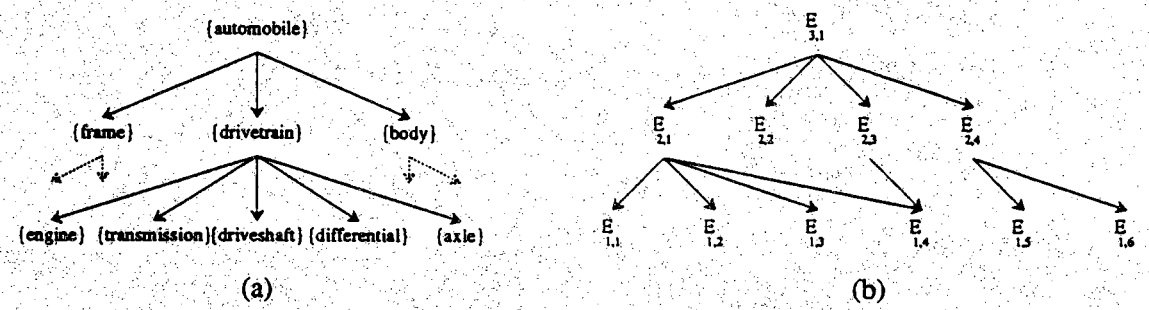


Figure 5.2 Panel (a) shows a hierarchical description of an automobile. Panel (b) demonstrates how a number of unidentified elements can be grouped into a part-of hierarchy.

elements on lower levels. For example, in this figure elements $E_{1,1}$ through $E_{1,4}$ are grouped to form element $E_{2,1}$ ($E_{i,j}$ denotes the j^{th} element on the i^{th} level of the hierarchy). Part-of hierarchies are a natural way to represent many types of objects. For example, this report contains a number of chapters each of which contains a number of sections. As we progress down this structure, we find that the sections can be broken down into paragraphs, sentences, clauses, words and letters. An automobile also can be represented hierarchically using a part-of hierarchy. Panel (a) of Fig. 5.2 is a simple example of how an auto can be broken down into its major assemblies (the frame, the body and the powertrain) and how each of these assemblies can be broken down into its main components. Of course, a part-of hierarchy that could be used to represent a real auto would be much more complex. Note that these hierarchies do not need to be strict; that is, an element can have more than a single parent if it is in more than one group. For example, many body components in a Unibody automobile also serve as part of the frame.

The structure of a part-of hierarchy can be used to aid in the determination of the identity of its elements. For example, in many cases, the label for an element will dictate the possible labels that its descendents can assume. For example, in Fig. 5.2, if elements $E_{1,1}$ through $E_{1,4}$ are grouped to form element $E_{2,1}$ and if $E_{2,1}$ is thought to be the drivetrain of an auto, then the possible labels for elements $E_{1,1}$ through $E_{1,4}$ would be

$$\Theta = \{\text{engine, transmission, driveshaft, differential, axle}\}$$

Thus panel (a) of Fig. 5.2 can be thought of as a hierarchical arrangement of the possible

labels that the elements of panel (b) can assume (i.e. their frames of discernment). If the hierarchy was not used to restrict certain possible labels from being included in an element's FOD, then the FOD might include all possible labels on the same level of the hierarchy. As it stands, the FOD for an element is determined by its parent's label-element and the children of its parent's label-element. Specifically, an element's FOD is defined to be the children of its parent's label-element.

Because an element's FOD is determined by its parent's label, the FOD for the element and all of its descendents must change if the parent's label changes -- a computationally intensive operation. Thus it is advantageous to incorporate new evidence on upper levels of the hierarchy before incorporating evidence on lower levels of the hierarchy. Because calculations on upper and lower levels of the hierarchy can be thought to correspond to checking global and local consistencies respectively, generating updating evidence for elements on the upper levels of the hierarchy before generating updating evidence for elements on lower levels corresponds to performing global consistency checks before local ones.

To further curtail the number of uncertainty calculations, elements are used to generate updating evidence only for their siblings. For example, only elements thought to be part of the auto's drivetrain would be used to generate updating evidence for other elements in the drivetrain. If the data were not arranged hierarchically, every element would be needed to generate updating evidence for every other element.

It should be mentioned that if the updating belief for a number of siblings is generated by noting the degree to which their labels are mutually compatible, then the updating evidence contained in their updating SEFs should not be incorporated into their belief functions until all of the updating SEFs are formed. If the incorporation of the updating SEFs is not delayed in this manner, then it is possible for the updating SEF for an element to be influenced by its belief in its own label. An element could provide updating evidence to itself if it was used to generate updating evidence in another element's label which in turn was used to provide updating evidence about the first element's label. Delaying the incorporation of updating evidence into elements' belief functions until all updating evidence has been generated prevents this from occurring.

5.5.1. Evidence Propagation Between Levels in the Hierarchy

Evidence from an element's siblings is not the only source of knowledge used to update its belief function. A mechanism also is provided for passing belief values between different levels of the hierarchy. This is done to satisfy the intuitive argument that says any evidence confirming an element's label also should provide evidence that its parent's label is correct. Disconfirming evidence on upper levels of the hierarchy also affects the belief functions of elements on the lower levels of the hierarchy. Furthermore, it is intuitively appealing to pass both confirmatory and disconfirmatory information up the hierarchy if all updating evidence for an element is generated by measuring its consistency with its siblings.

If the children of an element have consistent (compatible) labels, then these child-elements should provide evidence that the label given to the parent-element is correct. Likewise children with inconsistent labels provide evidence that their parent's label is incorrect. The updating SEF from the child with the largest belief† is used as the measure of the consistency of the labels of an element's children. Thus to propagate evidence up the hierarchy, the updating BPA, $m_{update}^{label} = M^{label}$, from the most believed child is accumulated not only into the child's belief function but also into the parent's belief function. Combining the updating SEF with an element's parent makes intuitive sense because all new evidence generated on a level comes from the (in)compatibility between elements on that level. The SEF from the most believed child is chosen as the measure of the consistency because that child can be though of as the main child of the group; if this element is labeled incompatibly with its siblings, then the group is said to be labeled inconsistently. Thus, by passing the updating SEF to each parent-element, new evidence is provided for those elements based on the consistency or the inconsistency of their descendents.

Evidence from an element cannot be applied directly to its parent because the FODs of an element and its parent are composed of different types of data elements. However, it will be shown that it is possible to build a FOD that can be used to update the belief functions of elements on a higher level of the hierarchy. Assume that the data is as

† The updating SEFs from all of the element's children are not combined together, as was done in earlier versions of PSEIKI, because these SEFs contain a large amount of redundant information. By propagating only one updating SEF up the hierarchy, the problem of combining dependent belief functions is avoided.

shown in panel (b) of Fig. 5.2 and that element $E_{1,1}$ is the child of element $E_{2,1}$ with greatest belief. Furthermore, assume that $E_{1,1}$ is labeled as the transmission and $E_{2,1}$ is labeled as the drivetrain. Because the confirmatory evidence for $E_{1,1}$'s label derived from its siblings arises from the consistency of the label with its sibling's labels, it may be considered as a weighted vote of confidence that $E_{2,1}$'s label is correct. Likewise, because the disconfirmatory evidence for $E_{1,1}$'s label derived from its siblings arises from the inconsistency of the label with its sibling's labels, it may be considered as a (weighted) vote of no confidence in $E_{2,1}$'s label. Thus, $m_{\underset{E_{1,1}}{update}}(\Theta)$ can be considered to be the amount of ignorance in $E_{2,1}$'s label. Using this rationale, an updating SEF for $E_{2,1}$ with the following non-zero probability masses may be defined as

$$m_{\underset{E_{1,1}\rightarrow E_{2,1}}{update}}(\{\text{drivetrain}\}) = m_{\underset{E_{1,1}}{update}}(\{\text{transmission}\}) \times SF_{propagate} \tag{5.47}$$

$$m_{\underset{E_{1,1}\rightarrow E_{2,1}}{update}}(\{\neg\text{drivetrain}\}) = m_{\underset{E_{1,1}}{update}}(\{\neg\text{transmission}\}) \times SF_{propagate}$$

$$m_{\underset{E_{1,1}\rightarrow E_{2,1}}{update}}(\Theta_{E_{2,1}}) = 1.0 - m_{\underset{E_{1,1}\rightarrow E_{2,1}}{update}}(\{\text{drivetrain}\}) - m_{\underset{E_{1,1}\rightarrow E_{2,1}}{update}}(\{\neg\text{drivetrain}\})$$

If it is assumed that $m_{\underset{E_{1,1}}{update}}$ is a valid SEF, then $m_{\underset{E_{1,1}\rightarrow E_{2,1}}{update}}$ also is a valid SEF. $SF_{propagate}$ is used to limit the amount of belief propagated up the hierarchy ($0.0 \leq SF_{propagate} \leq 1.0$). Depending on the application, it may be advantageous to limit the amount of evidence propagated up the hierarchy if the hueristically defined compatibility measures are not trusted completely. The total accumulated new belief for $E_{2,1}$ from its children $E_{1,1}$, ..., $E_{1,4}$ now can be expressed as

$$m_{\underset{E_{2,1}}{update}} = (m_{\underset{E_{1,1}\rightarrow E_{2,1}}{update}} \oplus \ldots \oplus m_{\underset{E_{1,4}\rightarrow E_{2,1}}{update}}) \tag{5.48}$$

Information is passed down the hierarchy only if it is disconfirmatory. This downward propagation of information takes the form of the reassignment of frames of discernment caused by the ancestor of an element having its label changed. In the previous example, this could happen if the hypothesized identity for $E_{2,1}$ is changed to be the frame of the auto. Using model information, the FOD for E_1 would be reassigned to

$$\Theta = \{\text{carraige, front suspension, rear suspension}\}$$

It should be mentioned that there are two cases that require special consideration. First, a data-element may have no siblings; in this case, since the element's consistency

can not be checked with its siblings, the only updating evidence that will be received about its label will be generated by checking the consistency of its children's labels. The other special case occurs when an element's label-element is an only child; in this case, there is only one member of the element's FOD. Therefore, the element's label can not be changed no matter how small the belief in this label becomes. Note that since the element has only one element in its FOD ($|\Theta|=1$), its belief function is a simple support function.

It should be noted that some of the evidence propagated up the hierarchy from an element's descendents depends on the element's belief function. The evidence is dependent on the belief function because the labels of an element's children (and hence the evidence focusing on those labels) depend on the label of the parent. Thus, in a roundabout manner, the propagated evidence depends on the parent's belief function. No investigation has been performed to determine the effect of the dependent belief functions. Some previous work addressing this topic can be found in [DubPra85], [DubPra86], [HunJay87], [Kyb87], [Sme76], and [Yen86].

5.6. Use of the Hierarchical Evidence Accumulation Scheme in PSEIKI

The evidence accumulation scheme introduced here originally was developed to aid in the matching of data-elements with model-elements by PSEIKI's labeler KS. In this application, the labeler KS uses the scheme to determine the identities of the elements on the data panel of the blackboard. Their possible identities are the elements on model panel. To illustrate how the scheme is used by PSEIKI, consider the example in Fig. 5.3. This figure shows the edge-level and face-level of the data on the blackboard. Model-data is shown in the left panel; in this frame edges E_A through E_D are grouped into face F_A. The right panel shows image-data; here edges E_1 through E_7 are grouped into face F_1[†].

As was mentioned in the previous section, the hierarchical nature of the matching task is used to increase the efficiency of the matching process by restricting the model-elements allowed to be members of an image-element's FOD. For example, in Fig. 5.3, if F_1 is matched with F_A, then the frame of discerment for edges E_1 through E_7 would be

[†] Note that in the following discussion the elements generated by the graphics source have capital letters as subscripts while elements derived from 2D vision data have numeric subscripts.

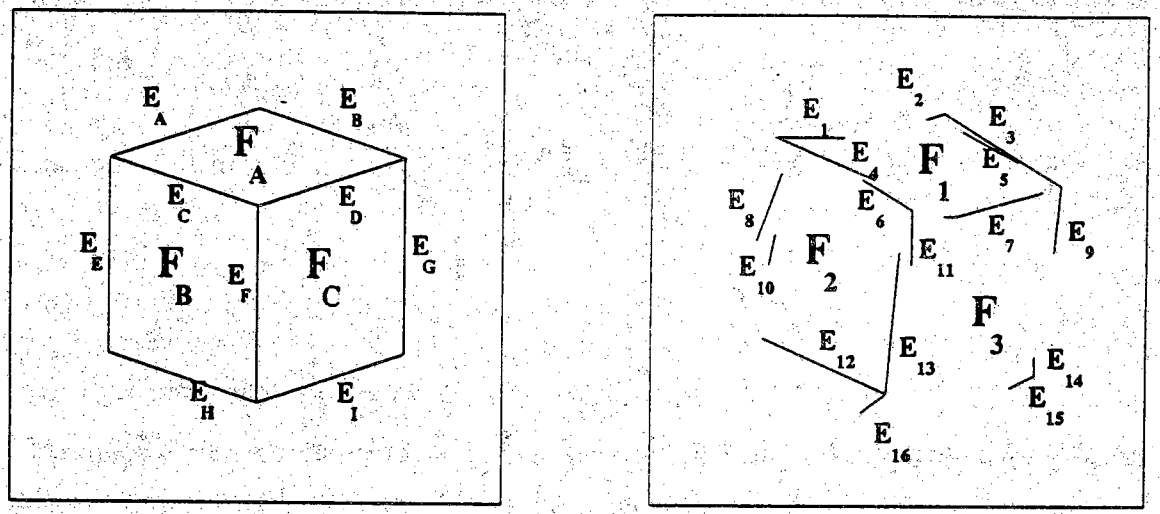


Figure 5.3 The left panel of this figure shows a simple example of model-elements derived from a solid modeling system; the right panel shows image-elements from 2D vision data. The figure is used to aid the textual explanation of how PSEIKI's labeler KS uses the evidence accumulation scheme introduced here. Note that the elements in this figure could represent only a small fraction of the data-elements on the blackboard panels.

$$\Theta = \{E_A, E_B, E_C, E_D\}$$

These model elements are allowed be members of the FODs for edges E_1 - E_7 because they are the children of their parent's model element, face F_A.

The hierarchical nature of the task is exploited further by checking the consistency of an element only with its siblings. In the previous example, the belief E_1's label would be updated only with evidence generated by noting its consistency with edges E_2 - E_7. These edges would be used to provide the updating evidence because they are grouped into face F_1 along with edge E_1. If the image-elements were not grouped hierarchically, then every edge would be needed to generate updating evidence in E_1's label. The method used to generate updating evidence for edges and faces based on their consistency with their siblings is discussed in chapter 6.

PSEIKI's labeler KS also propagates updating SEFs up the hierarchy in the previously discussed manner. For example, assume that edge E_1 has the greatest belief of all of face F_1's children. If E_1 has label E_A and face F_1 has label F_A, then the following updating SEF for F_1 can be created from the updating SEF for E_1

$$m_{\underset{E_1 \rightarrow F_1}{update}}(F_A) = m_{\underset{E_1}{update}}(E_A) \times SF_{edge-propagate}$$

$$m_{\underset{E_1 \rightarrow F_1}{update}}(\neg F_A) = m_{\underset{E_1}{update}}(\neg E_A) \times SF_{edge-propagate}$$

$$m_{\underset{E_1 \rightarrow F_1}{update}}(\Theta_{F_1}) = 1 - m_{\underset{E_1 \rightarrow F_1}{update}}(F_A) - m_{\underset{E_1 \rightarrow F_1}{update}}(\neg F_A)$$

The SEFs are propagated upwards for the reasons discussed earlier. Compatibly labeled siblings should provide confirmatory evidence about their parent's label; conversely, incompatibly labeled siblings should provide disconfirmatory evidence about their parent's label. The level-specific scaling factor, $SF_{edge-propagate}$, is used to limit the amount of information passed up the hierarchy ($0.0 \leq SF_{propagate} \leq 1.0$). As in the general scheme, changing the SEF for an element on an upper level of the hierarchy will force all of its descendents to change their FODs. The FODs are changed to satisfy the heuristic which states, for example, that the constituent edges of a mislabeled face also are most likely mislabeled.

5.7. Another Application of the Hierarchical Evidence Accumulation Scheme

It also is possible to use the hierarchical evidence accumulation scheme developed here in domains, other than computer vision, which are suitable for blackboard processing. The scheme is applicable to these domains because of their hierarchical nature. For example, the evidence accumulation scheme could be used in the domain for which the Hearsay-II [ErmHay80] blackboard system was developed: speech understanding.

Speech is represented hierarchically in the Hearsay-II system on the following 6 levels: *phrases*, *word-sequences*, *words*, *syllables*, *segments* and *parameters*. The lowest-level of the representation, the parameter level, breaks the speech waveform into five classes: silence, sonorant peak, sonorant nonpeak, fricative and flap. The next higher level, the segment level, is used to label the elements on the parameter level with phoneme-like labels. These labels are generated using statistical pattern recognition techniques and can assume 98 different values. Hearsay-II forms the elements on the higher levels of the hierarchy (the syllable, word, word-sequence and phrase levels) by

grouping compatible elements from the lower levels.

To apply the accumulation scheme to Hearsay-II's task, the statistically-based classifier still could be used to generate phoneme-like labels for the parameter elements. However, initial belief values for the segments' labels could be generated from the probabilities produced by the segment classifier. Updating evidence for the elements' labels then could be based on the compatibility between the elements and their siblings, as is done in PSEIKI. For example, on the word level of the blackboard, if an adjective is followed by a noun then the two should lend support to each other.

Updating evidence could be passed up the hierarchy as is done in PSEIKI (for example, evidence that a word is correct would also be evidence that its parent phrase is correct). Likewise, changing the label of an element on an upper level of the blackboard would cause all of its descendents to change their FODs.

CHAPTER 6

GEOMETRIC COMPUTATIONS FOR INITIAL AND UPDATING BELIEF FUNCTIONS

Chapter 5 showed how evidence is used to generate and update belief in a data-element's label; however, no mention was made of how that evidence is generated. This chapter will address the process of generating evidence to choose initial labels for elements and to update the belief in those labels. In PSEIKI, evidence about an element's label is generated by measuring how the degree to which an element meets geometric constraints between itself and other elements. These constraints take two general forms. Initially when matches are being formed, the constraints measure the similarity between an image element and model elements. After the initial matches are found and a label for the element has been determined, the constraints are used to measure the consistency between the element's label and the labels of its siblings in the hierarchy.

There are many techniques which PSEIKI could use to determine whether elements meet geometric constraints. Besl describes some general techniques for matching image data and model data at various levels of abstraction (points, curves, surfaces and volumes) using geometric constraints [Bes88]. Crowley and Ramparany take a different approach to the process of generating evidence based on geometric constraints; they model sensor readings as samples from a multivariate Gaussian distribution and use this assumption to calculate a "distance" from a feature measurement to its mean value [CroRam87]. They then estimate the belief in an entity based on the distance measured. Regardless of the method used to measure the degree to which the elements meet geometric constraints, the constraint measurements must be converted into belief functions. The method used in PSEIKI to convert raw confidence values into simple evidence functions is described in appendix B; the conversion method described there is only one

possible method that could be used.

In this chapter, three components of the evidence generation process will be explored. The first section of this chapter addresses the methods used to determine the set of all possible labels for an element (i.e. its FOD). The second section addresses the generation of initial labels based on the compatibility of data-elements with model-elements. The final portion of the chapter addresses the process of generating updating evidence for an element's label based on the compatibility between its label and its siblings' labels,

6.1. Determining an Element's Frame of Discernment

The first step in finding an element's label is determining its FOD. If an element has a parent, then the its FOD is defined to be the children of its parent's label-element, as described in chapter 5. For example, consider Fig. 6.1. If edges $\{E_1, ..., E_7\}$ on the data panel are grouped into face F_1 and F_1 is matched with F_A, then the FOD for each edge in the group would be $\Theta = \{E_A, E_B, E_C, E_D\}$.

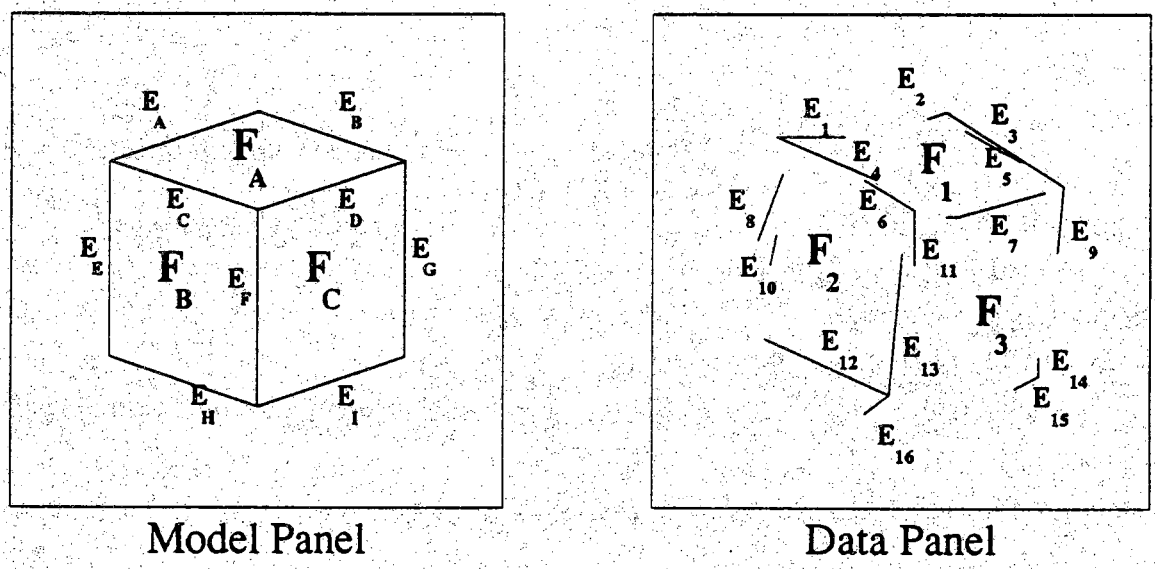


Figure 6.1 The left panel of this figure shows a simple example expected scene; the right panel shows image-elements from 2D vision data.

Note that the labels of elements on the upper levels of the blackboard should be determined before the labels of the lower level elements. The labels of the upper-level elements should be determined first because the FODs of the elements on the lower levels of the blackboard are determined by the labels of the elements on the upper levels. If an element has more than one parent, (for example, the two faces that an edge borders) then its FOD is defined as the union of each parent's label-element's children. Thus, whenever an upper-level element is given a label, its children's FODs are expanded to include the children of the parent's label element.

If an element has not been placed into a group and, therefore, has no parent†, then a different tack must be taken to form its FOD. In this case, the element's *expanded extent* is used to determine its FOD. The term extent is taken from computer graphics field [FolVan82] and is defined to be the minimum-size rectangle with edges parallel to the coordinate axis that contains an object. An extent is expanded by adding a border to each of its sides, as shown in Fig. 6.2. The size of the border around the extent, D_{max}, is set by the user and reflects the maximum expected misregistration between the image and the expected scene. Examples of the extents and expanded extents for an edge and a face are shown in Fig. 6.2. The FOD for an orphan element is defined to be the set of all model elements on the same level whose extents overlaps the orphan's expanded extent. Fig. 6.3 demonstrates the process of determining an orphan face's FOD. In cases (a) and (b) of this figure, F_A would be placed in F_1's FOD; however, in case (c) it would be excluded from the FOD because the two extents do not overlap. The data element's extent is expanded before applying the criterion to guarantee that the extent of a small data element will overlap with its true correspondent on the model panel. Fig. 6.4 shows how the same procedure is used to determine the FOD for an orphan edge. As in the previous example, in cases (a) and (b), edge E_A would be placed in E_1's FOD because its extent (shown as dashed boxes) overlaps with edge E_1's expanded extent. Conversely, it would be excluded in case (c) because the two extents do not overlap. At the edge level, expanded extents are used because two edges can be arbitrarily close without having overlapping extents (for example, if they are both parallel to the same coordinate axis).

† Elements with no parents are said to be *orphaned.*

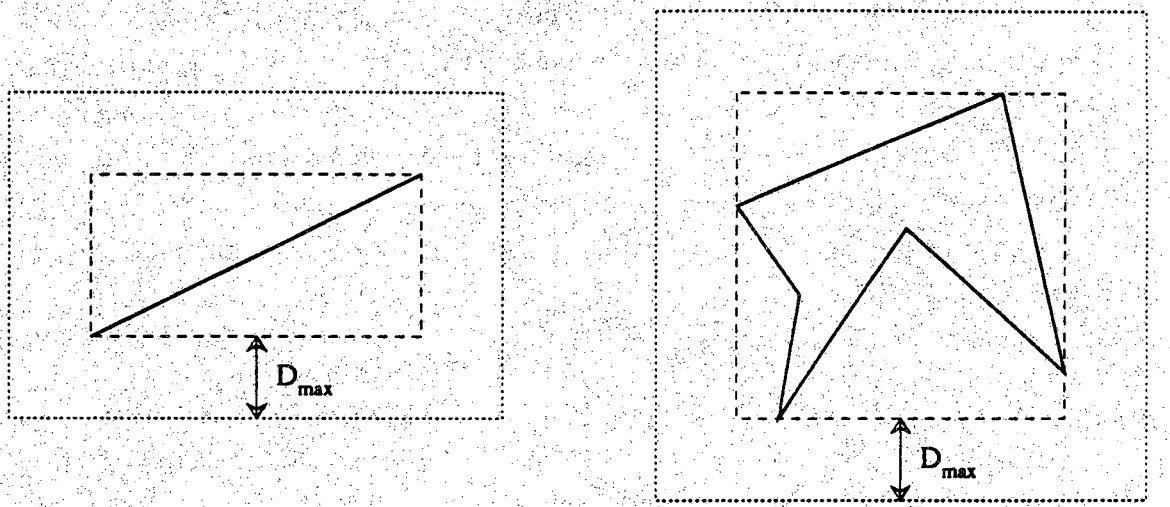


Figure 6.2 This figure shows the extent for an arbitrary face and an arbitrary edge. The objects in this figure are drawn using solid lines, their extents are the dashed boxes with heavy lines and the expanded extents are the dashed boxes with light lines.

6.2. Computing Initial Belief Functions for Data Elements

As described in chapter 5, an element's initial belief function is computed by combining simple evidence functions (SEFs), each of which is focused on a singleton proposition and its compliment. In PSEIKI, the SEF for a model element is determined by using the metaphor that the model element is an expert which indicates how much belief it places in its similarity to the data element. The degree of similarity between the data and model elements is obtained by measuring the degree to which constraints between the image-element and model-elements are met using level-specific metrics. The output of the metrics range from 0.0 to 1.0 to facilitate the conversion of the measurements into SEFs using the technique presented in appendix B. In other words, the expert θ_i yields the following information about data element ψ_j

$$M^{\theta_i}(\theta_i) = \text{similarity\_metric}(\psi_j, \theta_i)$$

$$M^{\theta_i}(\neg\theta_i) = \text{dissimilarity\_metric}(\psi_j, \theta_i)$$

$$M^{\theta_i}(\Theta) = 1 - M^{\theta_i}(\theta_i) - M^{\theta_i}(\neg\theta_i)$$

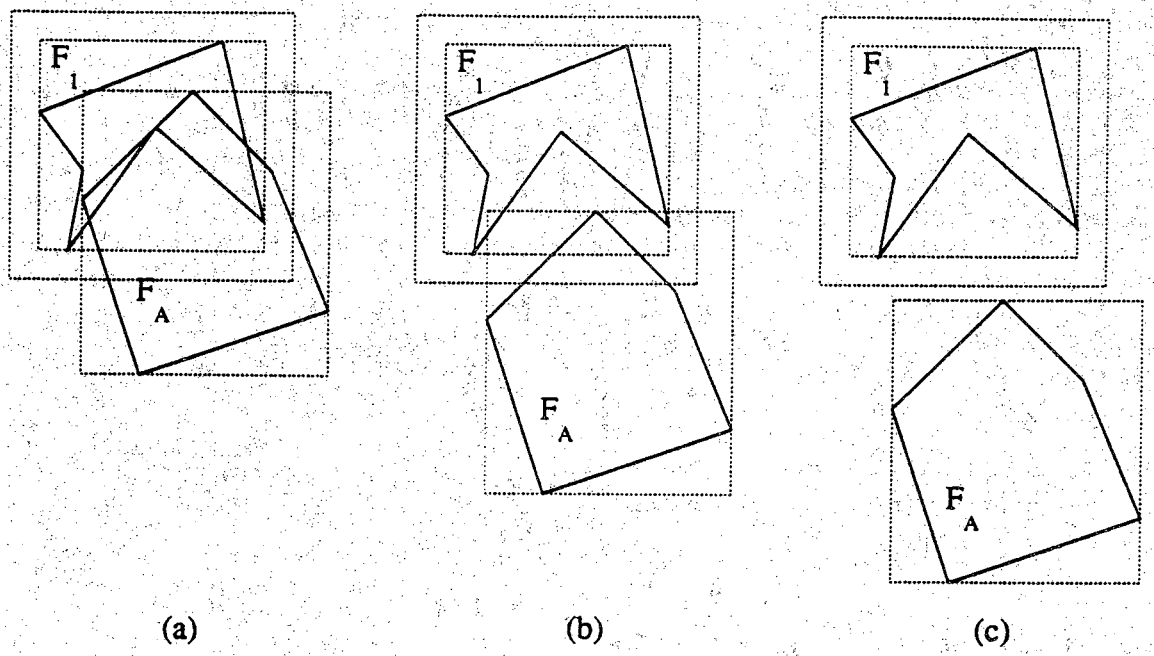


Figure 6.3 This figure demonstrates how extents are used to determine an orphan element's FOD. In this figure, face F_A would be placed in F_1's FOD in cases (a) and (b) because their extents (shown as dashed boxes) overlap with face F_1's expanded extent. Conversely, it would be excluded in case (c) because the two extents do not overlap. The same procedure is used to determine the FOD for an orphan element on the object level.

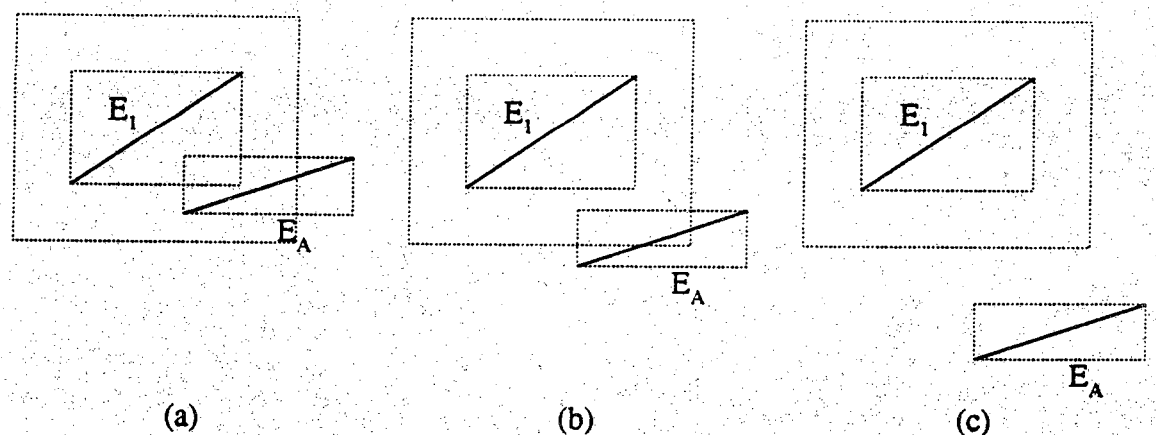


Figure 6.4 This figure demonstrates how extents are used to determine an edge's FOD. In this figure, edge E_A would be placed in E_1's FOD in cases (a) and (b) because their extents (shown as dashed boxes) overlap with edge E_1's expanded extent. Conversely, it would be excluded in case (c) because the two extents do not overlap.

After the SEFs for all members of an element's FOD have been computed, the element's label is determined using the procedure described in chapter 5.

6.2.1. Computing Initial Belief Functions for Face-Elements and Object-Elements

When determining the initial label of a face element or an object element, PSEIKI's labeler KS tries to pick the model element from the data element's FOD whose shape is most like the data element's shape and whose centroid is closest to the data element's centroid. Thus, to compute the data element's SEF focused on a particular model element, the shape of the data element is compared with the shape of the model element. The shapes are compared by translating the data element so that the centroid of its convex hull is aligned with the centroid of the model element's convex hull. After the centroids have been aligned, the percent of overlap between the data element and the model element is measured. The percentage of overlap between two elements is defined to be the area of the intersection of the data element's convex hull with the model element's convex hull divided by the area of their union. Fig. 6.5 shows how the percentage of overlap is measured. Panels (a) and (b) show the model and data elements, respectively; the solid lines denote the actual elements, the dashed line denote their convex hulls. The formula used to compute the percentage of overlap is shown below.

$$\text{overlap}(F_{\text{model}}, F_{\text{image}}) = \frac{\text{Area}_{\text{intersection}}}{\text{Area}_{\text{union}}}$$

Convex hulls are used in these computations because the edges forming a face are not guaranteed to form a closed boundary. Thus, in general, it is not always possible to find the border of an arbitrary collection of edges. The method used to determine the convex hulls is described in [Sed84]. It is a simple matter to find the intersection of the two hulls because their intersection also is guaranteed to be convex. After the area of the intersection of the two hulls has been found, the area of their union can be computed by summing the area of the two hulls and subtracting the area of their intersection.

The evidence in the SEF also depends on the relative location of the data element with respect to the model element. The locations are compared by measuring the distance between the centroids of their convex hulls. Combining the shape and distance metrics yields the final expected-scene compatibility and incompatibility metrics for the face and object levels of the blackboard.

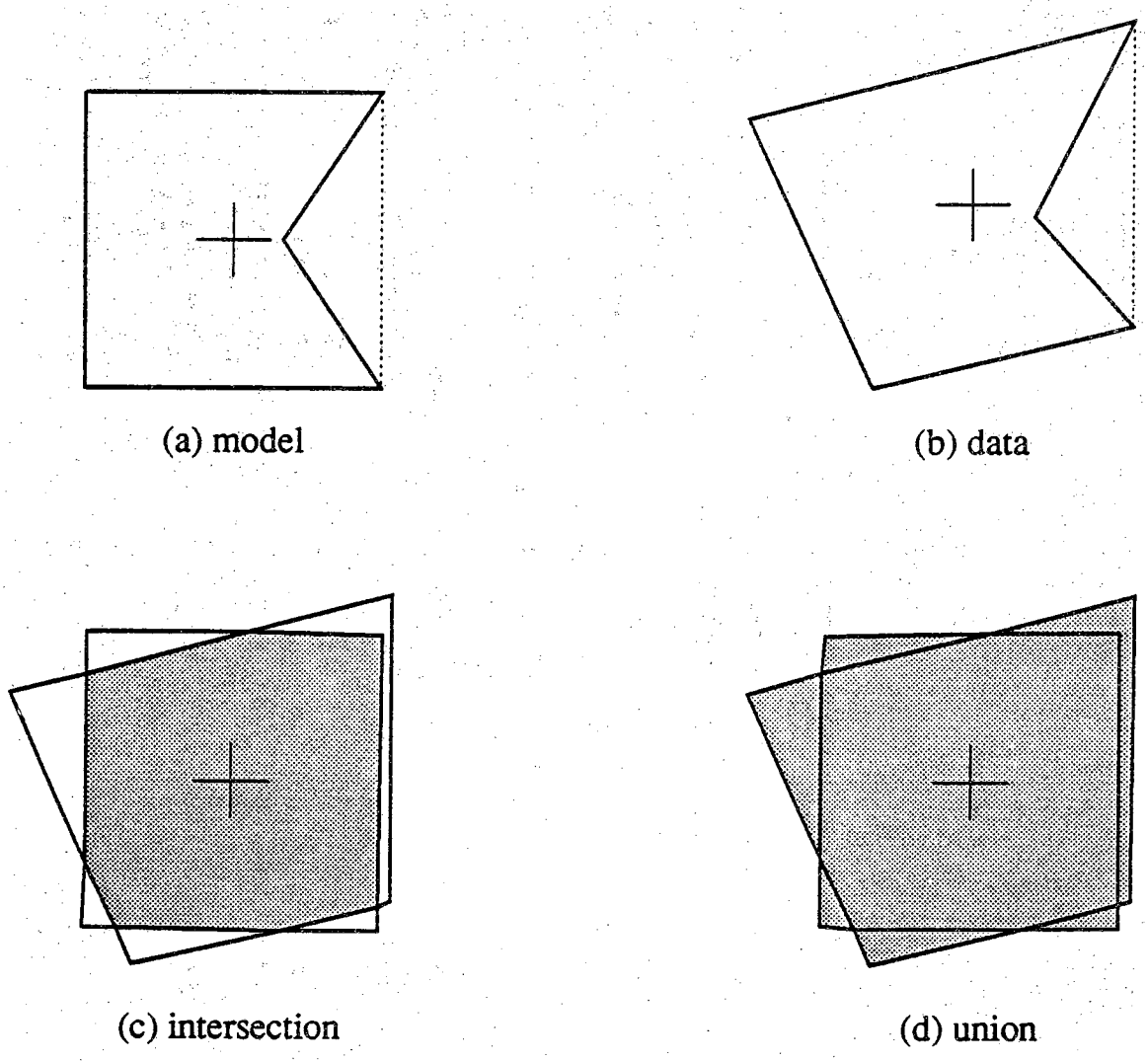


Figure 6.5 This figure demonstrates how the percentage of overlap between a data element and one of the model elements is computed. Panels (a) and (b) show the model and data elements (shown as solid lines), their convex hulls (shown as dashed lines) and centroids (shown as crosses). Panels (c) and (d), respectively, show the area of the intersection and union of their convex hulls after their centroids have been aligned.

$$ES\_compat(F_{model}, F_{image}) = overlap(F_{model}, F_{image}) \times \left[1 - \frac{D_{centroids}}{D_{max}}\right]^2 \times SF_{initial}$$

$$ES\_incompat(F_{model}, F_{image}) = \left[1 - overlap(F_{model}, F_{image})\right] \times \left[\frac{D_{centroids}}{D_{max}}\right]^2 \times SF_{initial}$$

$D_{centroids}$ is the distance between the centroid of the data element's convex hull and the centroid of the model element's convex hull. D_{max} is determined by the maximum expected misregistration and is set equal to the amount that the elements' extents are expanded when their FODs were determined (as described in section 6.1). If the distance between the two centroids is greater than D_{max}, then $D_{centroids}$ is set equal to D_{max}. The distance metric's exponent causes the metric's value to decrease rapidly with increasing distance to the model element; this sharp falloff increases the ability of the metric to discriminate between closely located model elements. $SF_{initial}$ is a level-specific scaling factor. It is used to restrict the amount of evidence provided by the hueristically defined metrics ($0.0 \leq SF_{initial} \leq 1.0$).

To construct the SEF, the procedure described in appendix B is used to convert the values determined by the (in)compatibility metrics into a belief function. Since *ES_compat()* and *ES_incompat()* are guaranteed to sum to less than one, the following SEF is produced by the conversion process.

$$M^{F_{model}}(F_{model}) = ES\_compat(F_{model}, F_{image})$$

$$M^{F_{model}}(\neg F_{model}) = ES\_incompat(F_{model}, F_{image})$$

$$M^{F_{model}}(\Theta) = 1 - M^{F_{model}}(F_{model}) - M^{F_{model}}(\neg F_{model})$$

To determine a face-element's label, PSEIKI's labeler KS uses the above procedure to compute a SEF for each member of its FOD. For example, if face F_1's FOD was determined to be

$$\Theta = \{F_A, F_B, F_C, F_D\}$$

then the formulas might produce the following *ES_compat()* and *ES_incompat()* measurements,

$$ES\_compat(F_A, F_1) = 0.43$$

$$ES\_incompat(F_A, F_1) = 0.23$$

$$ES\_compat(F_B, F_1) = 0.11$$

$$ES\_incompat(F_B, F_1) = 0.41$$

$$ES\_compat(F_C, F_1) = 0.73$$

$$ES\_incompat(F_C, F_1) = 0.03$$

$$ES\_compat(F_D, F_1) = 0.56$$

$$ES\_incompat(F_D, F_1) = 0.26$$

The following four SEFs result from applying the conversion process described in appendix B to the above data.

$$M^{F_A}(F_A) = 0.43 \qquad M^{F_B}(F_B) = 0.11$$

$$M^{F_A}(\neg F_A) = 0.23 \qquad M^{F_B}(\neg F_B) = 0.41$$

$$M^{F_A}(\Theta) = 0.34 \qquad M^{F_B}(\Theta) = 0.48$$

$$M^{F_C}(F_C) = 0.73 \qquad M^{F_D}(F_D) = 0.56$$

$$M^{F_C}(\neg F_C) = 0.03 \qquad M^{F_D}(\neg F_D) = 0.26$$

$$M^{F_C}(\Theta) = 0.24 \qquad M^{F_D}(\Theta) = 0.18$$

The procedure described in chapter 5 can then be used to determine the belief in the singleton propositions.

$$Bel(F_A) = 0.1318$$

$$Bel(F_B) = 0.0236$$

$$Bel(F_C) = 0.4795$$

$$Bel(F_D) = 0.2193$$

With these belief values, F_C would be assigned as the element's label with belief 0.48.

6.2.2. Computing Initial Belief Functions for Edge-Elements

When determining the initial label of an edge element, PSEIKI's labeler KS tries to match a data edge with the model edge in its FOD that lies closest to the line defined by

the edge. To find the match partner of a data edge, the KS measures the degree of "collinearity" and "noncollinearity" between the edge and all the model edges in its FOD; it then chooses as the match partner the model edge with which the data edge is most collinear. The belief assigned to the match depends on the degree of collinearity between the two edges.

If the edge is an orphan, then the following formulas are used to measure the collinearity and noncollinearity between an edge detected in the image and an edge from the expected scene.

$$\text{ES\_edge\_compat}(E_i, E_j) = \left[\frac{D_{max} - D_{perp}}{D_{max}}\right]^2 \times \left[\frac{D_{max} - D_{par}}{D_{max}}\right]^2 \times \cos^2(\theta) \times SF_{edge-initial}$$

$$\text{ES\_edge\_incompat}(E_i, E_j) = \left[\frac{D_{perp}}{D_{max}}\right]^2 \times \left[\frac{D_{par}}{D_{max}}\right]^2 \times \sin^2(\theta) \times SF_{edge-initial}$$

In these formulas, E_i is the model edge and E_j is the data edge. D_{perp} is the perpendicular distance from the middle of E_j to the line defined by E_i. D_{par} is the misregistration along the direction of E_i; if point Q falls on E_i, then D_{par} is set to zero. D_{max} is the same value used when determining the label of face elements and object elements, and θ is the acute angle between the segments (see Fig. 6.6). $SF_{edge-initial}$ is the edge-level initial evidence scaling factor ($0 \leq SF_{edge-initial} \leq 1$). As can by seen from these formulas, the metrics contain three parts (not including the scale factor)†. First, the D_{perp} component of the metrics measures the distance the data edge lies from the model edge in the perpendicular direction. If the center of the data edge lies close to the line defined by the model edge, then this component of the metrics will indicate that the edges are compatible. Second, the D_{par} component of the metrics measures the distance the data edge is displaced from the model edge along the line defined by the model edge. If the data edge is displace too far along the line, then this component of the metrics will indicate that the edges are incompatible. Third, the θ component of the metrics measures the angle

† [KreMik89] describes a detailed investigation of the edge-level metrics for a previous version of PSEIKI. In this investigation, they describe the affect the metrics had on the belief values attached to labels for various data/model edge configurations.

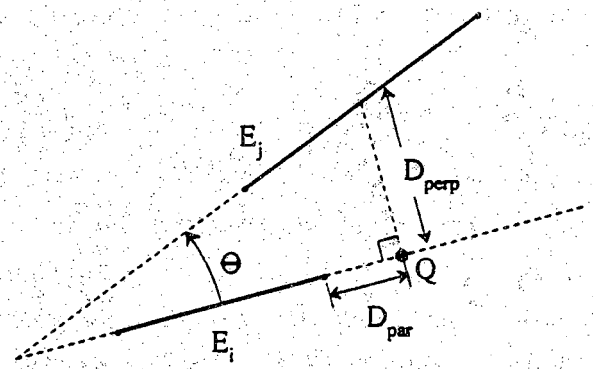


Figure 6.6 This figure shows geometry used for the edge-level constraints.

between the edges. If the edges are almost parallel, then this component of the metrics will indicate that the edges are compatible.

If the edge is child of a previously labeled face, then the parent face (and all of its edges) are transformed before applying the *ES_edge_compat()* and *ES_edge_incompat()* metrics. In the current version of PSEIKI, the parent face is translated such that its centroid aligns with its model face's centroid. At the present time, there is no translational component to the transformation applied to the parent face. No rotation is required in the mobile robot context because vertical lines remain approximately vertical. However, if PSEIKI was applied to other domains, a more general transformation including a rotational component could be implemented (for example, by aligning the major axes of the two faces). Fig. 6.7 demonstrates the reason for translating the face before the belief functions of its child edges are initialized. Panels (a) and (b) of this figure show a simple expected scene and observed image; notice that a simple rigid motion was applied to the expected scene to derive the observed image. If the face was not translated, as is shown in panel (c), then both E_1 and E_3 would be labeled as E_C. Edge E_1 would be labeled incorrectly because model edge E_C is the closest of the model edges. However, when the face is translated, both E_1 and E_3 receive the correct labels, E_A and E_C, respectively.

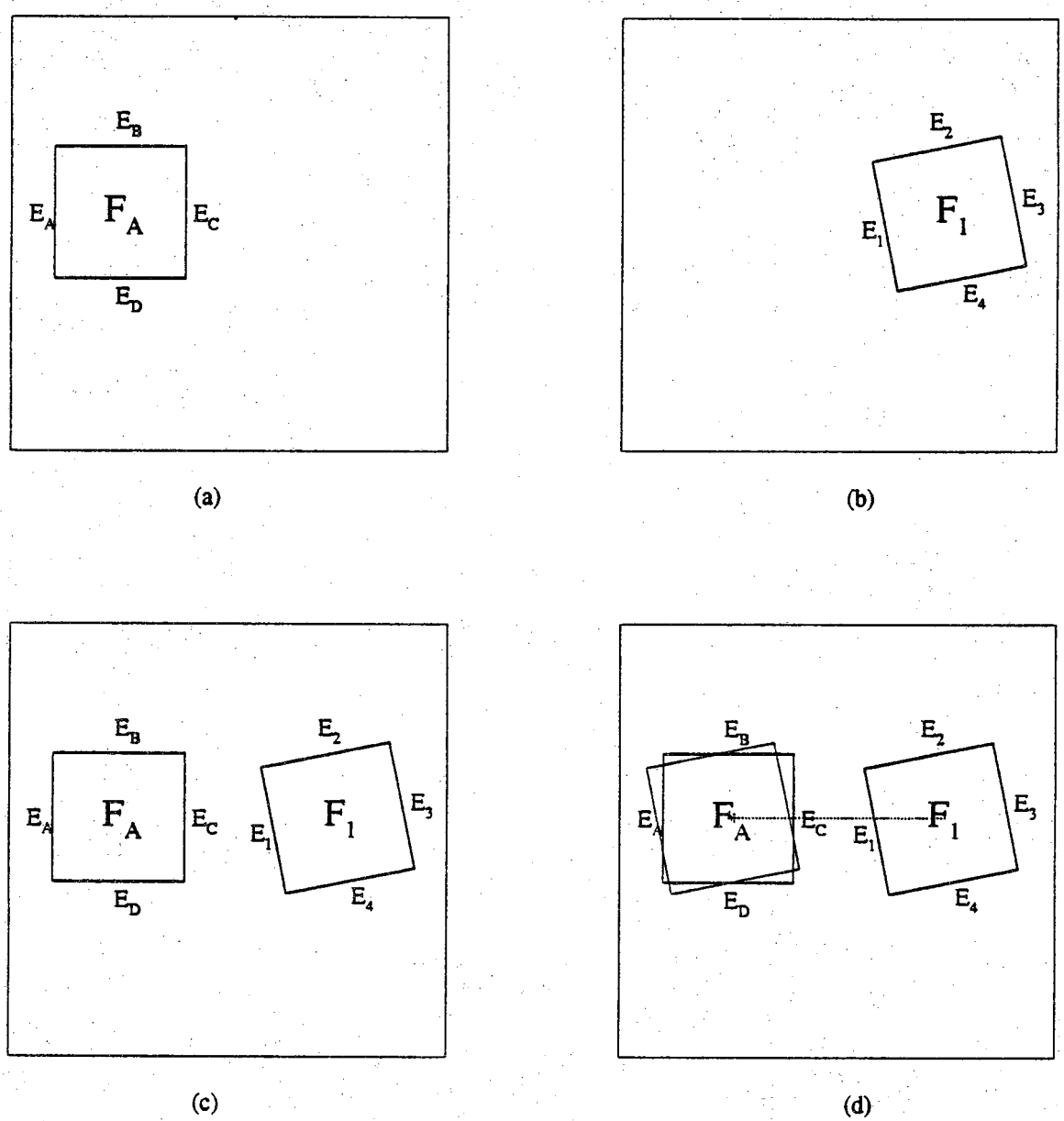


Figure 6.7 This figure demonstrates the reason for translating a parent face before the edge-level compatibility metric is applied.

The values produced by these metrics sum to less than one; thus, they can be converted into the following SEF.

$$M^{E_j}(E_j) = ES\_edge\_compat(E_i, E_j)$$

$$M^{E_j}(\neg E_j) = ES\_edge\_incompat(E_i, E_j)$$

$$M^{E_j}(\Theta) = 1 - M^{E_j}(E_j) - M^{E_j}(\neg E_j)$$

6.3. Computing Updating Belief Functions for Data Elements

After the initial matches are established between elements, the labeler KS provides updating evidence about the validity of the an element's label based on the label's consistency with the element's siblings' labels. In this phase, the updating SEFs for a data element are determined by using the metaphor that the each of the element's siblings are experts which generate updating evidence about the validity of the element's label. This evidence is based on the degree to which the two elements meet the geometrical relationships exhibited by their label elements. In general, two metrics are used to provide measures of *compatibility* and *incompatibility* between the element and its siblings. The compatibility metric measures the degree to which the geometric relationships are met, and is used to provide confirmatory evidence that the element's label is correct. Conversely, the incompatibility metric measures the degree to which the relationships are not met, and provides disconfirmatory evidence about the element's label. Both metrics range between 0.0 and 1.0 to facilitate conversion of their values to an updating SEF. The metrics are scaled by the belief in the label of the element providing the evidence; this scales the amount of information provided by an expert in proportion to the expert's "reliability." In other words, the element ψ_i with label θ_i can be used to update the belief in its sibling ψ_j with label θ_j

$$M^{\theta_i}(\theta_i) = \text{compatibility}(\psi_i, \psi_j; \theta_i, \theta_j) \times Bel_{\psi_j}(\theta_j)$$

$$M^{\theta_i}(\neg\theta_i) = \text{incompatibility}(\psi_i, \psi_j; \theta_i, \theta_j) \times Bel_{\psi_j}(\theta_j)$$

$$M^{\theta_i}(\Theta) = 1 - M^{\theta_i}(\theta_i) - M^{\theta_i}(\neg\theta_i)$$

where $Bel_{\psi_j}(\theta_j)$ is the belief attached to data element ψ_j's label, θ_j. It is illustrative to examine how one element can be used to update the belief in another element's label when both have the same label. When this process is understood, the case in which two

elements have different labels follows naturally.

6.3.1. Computing Updating Belief Functions for Face-Elements and Object-Elements with the Same Label

The (in)compatibility metrics for face-elements and object-elements are called *collocate()* and *noncollocate()*. These two metrics are designed to measure how close two elements are to each other by measuring the distance between their centroids. If two faces or objects with the same label are close enough to each other, then they should lend mutual support to the belief in each other's label (because the image preprocessor most likely incorrectly split them). The compatibility metric between two elements, collocate(F_1, F_2), is defined as

$$\text{collocate}(F_1, F_2) = \frac{D_{max} - D_{centroid}}{D_{max}}$$

where $D_{centroid}$ is the distance between the centroids of the two elements' convex hulls. For the computation of updating evidence, D_{max} is set in a manner different from that described in previous sections; currently, it is set to the length of the diagonal of F_1's extent. Setting D_{max} in this manner is justified by the rationale that the maximum allowable distance between two data-elements with the same label should be a function of the sizes of the data-elements. Similarly, noncollocate(F_1, F_2), the incompatibility metric is defined as

$$\text{noncollocate}(F_1, F_2) = \frac{D_{centroid}}{D_{max}}$$

Because the (in)compatibility measures are defined heuristically, it usually is advantageous to limit the amount of evidence that they can provide. This is accomplished by scaling the measures by a level-specific scale factor $SF_{updating}$, $(0.0 \leq SF_{updating} \leq 1.0)$. Thus the (in)compatibility measures for the face-level can be defined as:

$$\text{compatibility}(F_i, F_j) = \text{collocate}(F_i, F_j) \times SF_{face-updating}$$

$$\text{incompatibility}(F_i, F_j) = \text{noncollocate}(F_i, F_j) \times SF_{face-updating}$$

Once the (in)compatibility between the two faces has been determined, the technique described in appendix B can be used to convert them into an updating SEF. For example, assume that faces F_1 and F_2 exhibit maximal beliefs for the same model face, F_A, and

that the labeler KS is using F_2 to update the belief of F_1's label. To do so, the labeler applies the *collocate()* and *noncollocate()* metrics to F_1 and F_2. If the results of the (in)compatibility measurements are

$$\text{compatibility}(F_2, F_1) = 0.8$$

$$\text{incompatibility}(F_2, F_1) = 0.1$$

the belief in F_2's label (say, for example, 0.8) can be used to create an updating SEF for F_1 as follows:

$$M^{F_A}_{F_2 \to F_1}(F_A) = Bel_{F_2}(F_A) \times \text{compatibility}(F_2, F_1) = 0.64$$

$$M^{F_A}_{F_2 \to F_1}(\neg F_A) = Bel_{F_2}(F_A) \times \text{incompatibility}(F_2, F_1) = 0.08$$

$$M^{F_A}_{F_2 \to F_1}(\Theta) = 1.0 - M^{F_A}_{F_2 \to F_1}(F_A) - M^{F_A}_{F_2 \to F_1}(\neg F_A) = 0.28$$

Where $M^{\theta_k}_{\psi_i \to \psi_j}$ is the SEF containing updating evidence for ψ_j focusing on θ_k based on ψ_j's compatibility with its sibling, ψ_i.

6.3.2. Computing Updating Belief Functions for Edge-Elements with the Same Label

Collinearity() and *noncollinearity()* are the metrics used to determine the (in)compatibility between two edges with the same label (they are related to the *ES_edge_compat()* and *ES_edge_incompat()* measures used to establish initial matches). That is, if E_i and E_j are edges in the data panel and have the same label, then collinearity(E_i, E_j) is the measure of compatibility between them. Collinearity is defined as

$$\text{collinearity}(E_i, E_j) = \frac{D_{max} - D_{perp}}{D_{max}} \times \cos(\theta)$$

where θ is the acute angle between the two edges and D_{perp} the perpendicular distance from the middle of E_j to the line defining E_i (see Fig. 6.6). D_{max}, the maximum allowable value for D_{perp}, is set equal to the length of E_i. Again, this is done to scale, by an element's size, the evidence that the metric can provide.

Likewise, the incompatibility between two edges, can be measured by calculating the *noncollinearity()* between them. Noncollinearity is defined as

$$\text{noncollinearity}(E_i, E_j) = \frac{D_{perp}}{D_{max}} \times \text{scale}(E_i) \times \sin(\theta)$$

Scale(E_i) is used to limit the amount of disconfirmatory evidence generated by small edges which may be due to noise. It assumes a value of 0.0 for small (2 pixel long) edges and increases linearly to 1.0 for edges up to a prespecified length; it assumes a value of 1.0 for all edges greater than the prespecified length. The amount of belief given to small noisy edges is limited to prevent them from providing a large amount of disconfirmatory evidence to large, correctly labeled edges. As is done on the face-level, the values provided by these metrics are multiplied by a level-specific scale factor, $SF_{edge-update}$, to limit their evidence.

$$\text{compatibility}(E_i, E_j) = \text{collinearity}(E_i, E_j) \times SF_{edge-updating}$$

$$\text{incompatibility}(E_i, E_j) = \text{noncollinearity}(E_i, E_j) \times SF_{edge-updating}$$

Once the (in)compatibility between the edges has been determined, the technique described in appendix B is used to construct a SEF. For example, assume that E_1 and E_2 exhibit maximal beliefs for the same model edge, E_A. If the labeler KS is using E_2 to update the belief of E_1's label, then it would apply the collinearity and noncollinearity metrics to them. If the results of the (in)compatibility measurements are

$$\text{compatibility}(E_2, E_1) = 0.5$$

$$\text{incompatibility}(E_2, E_1) = 0.1$$

the belief in E_2's label (say, for example, 0.7) can be used to create an updating SEF for E_1 as follows:

$$M^{E_A}_{E_2 \to E_1}(E_A) = Bel_{E_2}(E_A) \times \text{compatibility}(E_2, E_1) = 0.35$$

$$M^{E_A}_{E_2 \to E_1}(\neg E_A) = Bel_{E_2}(E_A) \times \text{incompatibility}(E_2, E_1) = 0.07$$

$$M^{E_A}_{E_2 \to E_1}(\Theta) = 1.0 - M^{E_A}_{E_2 \to E_1}(E_A) - M^{E_A}_{E_2 \to E_1}(\neg E_A)$$

6.3.3. Computing Updating Belief Functions for Elements with Different Labels

If two elements correspond to different model-elements, a rigid motion transformation is applied to one of them before the computation of the (in)compatibility metrics. This has the effect of enforcing relational constraints between the two data-elements. For

example, if edges E_1 and E_3 are thought to correspond to model edges E_A and E_B, respectively, then the measure of compatibility between E_1 and E_3 would be defined as

$$\text{compatibility}(E_3, E_1) = \text{collinearity}(E_3, T_{E_A \rightarrow E_B}(E_1)) \times SF_{\text{edge-updating}}$$

where $T_{E_A \rightarrow E_B}$ is the rigid motion transformation that makes model edge E_A collinear with model edge E_B.

Fig. 6.8 can be used to aid in the explanation of the definition of the transformation. First, for a given pair of non-parallel edges, the vertices on the convergent side are found; the convergent side of the two edges is the side on which they would meet if extended. The transformation $T_{E_A \rightarrow E_B}$ is accomplished by rotating edge E_A about its convergent vertex through an angle that makes the edges parallel; subsequently, E_A is translated so that the two convergent vertices coincide. Performing this transformation forces model-elements to be compatible; in other words,

$$\text{collinearity}(E_B, T_{E_A \rightarrow E_B}(E_A)) = 1.0$$

Note that the definition of the transformation is not well defined. There are two transformations (depending on the direction that edge E_A is rotated) that can be used to make the two model edges collinear. The first transformation "unfolds" the two model edges by forcing the angle between them to be 180 degrees; this type of transformation is shown in panel (a) of Fig. 6.8. The other type of transformation "collapses" the two edges onto each other by forcing the angle between them to be 0 degrees as shown in panel (b) of Fig. 6.8. It is impossible to determine completely from the geometry of the model edges which transformation will be needed to make two data edges collinear; the transformation also depends on the direction that the image is misregistered from the expected scene. Therefore, the transformation that should be used to make two edges collinear must be determined at runtime. PSEIKI's labeler KS computes the collinearity of the two edges using both transformations and uses the transformation that results in the largest collinearity measurement. The same transformation then is used when applying the noncollinearity metric.

Consider, as an example, how the relational constraints are checked by transforming elements and measuring their (in)compatibility. Assume that edge E_3 in Fig. 6.1 is being used to provide updating evidence about the label of edge E_1. Furthermore, assume that edge E_1 has label E_A and edge E_3 has label E_B. To measure the extent to which the geometrical relationship between E_1 and E_3 is the same as the one between E_A and E_B,

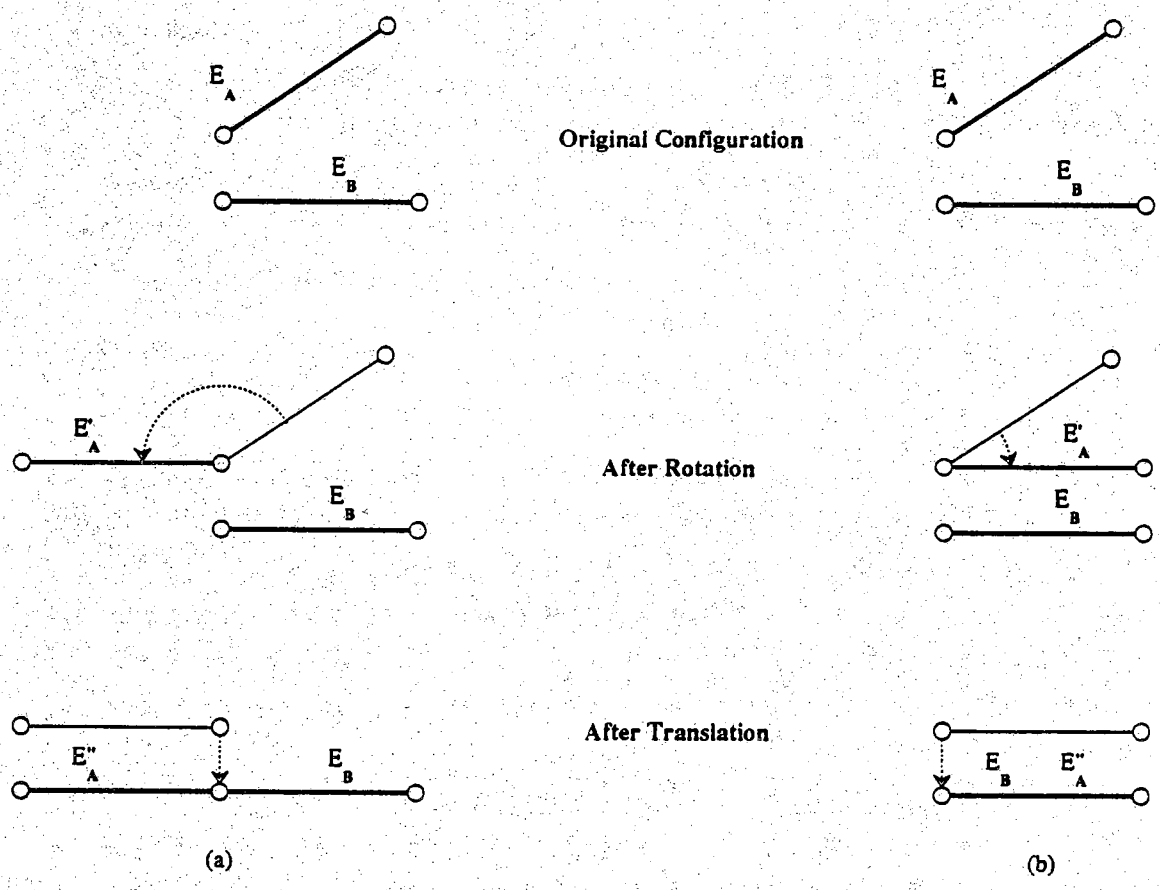


Figure 6.8 This figure shows the rigid motion transformation that makes two model-elements collinear. Panel (a) shows the transformation created by "unfolding" the two edges. Panel (b) shows the transformation created by "collapsing" the two edges.

the labeler carries out the following (in)compatibility computations:

$$\text{compatibility}(E_3, E_1) = \text{collinearity}(E_3, T_{E_A \rightarrow E_B}(E_1)) \times SF_{\text{edge-updating}}$$

$$\text{incompatibility}(E_3, E_1) = \text{noncollinearity}(E_3, T_{E_A \rightarrow E_B}(E_1)) \times SF_{\text{edge-updating}}$$

where $T_{E_A \rightarrow E_B}$ is the transformation that makes the model edges E_A and E_B collinear and results in the greatest measured collinearity between E_3 and the transformed version of edge E_1. Clearly, $\text{compatibility}(E_3, E_1) = 1.0$ implies that the geometrical relationship between E_1 and E_3 in the data is exactly the same as between E_A and E_B in the model (in this case, $\text{incompatibility}(E_3, E_1) = 0.0$). If the compatibility calculations yielded the following results:

$$\text{compatibility}(E_3, E_1) = 0.5$$

$$\text{incompatibility}(E_3, E_1) = 0.15$$

and the belief in E_3's label was 0.95 then the following SEF could be defined by using the (in)compatibility measures and the belief in E_3's label.

$$M^{E_A}_{E_3 \rightarrow E_1}(E_A) = Bel_{E_3}(E_B) \times \text{compatibility}(E_3, E_1)$$

$$= 0.95 \times 0.5$$

$$= 0.475$$

$$M^{E_A}_{E_3 \rightarrow E_1}(\neg E_A) = Bel_{E_3}(E_B) \times \text{incompatibility}(E_3, E_1)$$

$$= 0.95 \times 0.15$$

$$= 0.1425$$

$$M^{E_A}_{E_3 \rightarrow E_1}(\Theta) = 1.0 - M^{E_A}_{E_3 \rightarrow E_1}(E_A) - M^{E_A}_{E_3 \rightarrow E_1}(E_A)$$

$$= 1.0 - 0.475 - 0.1425$$

$$= 0.3825$$

The same technique of checking relational constraints by measuring the consistency of transformed data elements can be used on the elements residing on the face level. That is, the (in)compatibility between a face element and one of its siblings with a different label can be measured by applying the (non)collocate metrics between a transformed version of the face and its sibling. The following procedure is used to

measure the degree to which a face element and one of its siblings with a different label meet the relational constraints defined by their respective label elements. First, the transformation needed to make the centroids of both elements' label-element's coincident is computed. This transformation is applied to the face whose label is being updated. After the face has been transformed, the (non)collocate metrics are applied to measure the compatibility of the transformed face and its sibling. Because the metrics used to calculate the (in)compatibility between face-elements use only the distance between centroids for their computations, only a translational transformation is required. For example, if face F_3 and F_5 are thought to correspond to model faces F_B and F_C, respectively, then the measure of (in)compatibility between F_3 and F_5 would be defined as

$$\text{compatibility}(F_3, F_5) = \text{collocate}(F_3, T_{F_C \rightarrow F_B}(F_5)) \times SF_{\text{face-updating}}$$

$$\text{incompatibility}(F_3, F_5) = \text{noncollocate}(F_3, T_{F_C \rightarrow F_B}(F_5)) \times SF_{\text{face-updating}}$$

where the transformation $T_{F_i \rightarrow F_j}$ translates the centroid of face F_i's convex hull until it is coincident with the centroid of face F_j's convex hull. If the two data faces exhibit the same spatial relationship shown by their label elements, then the transformation will make the data faces' centroids coincident. In this case, the metrics will indicate that the two data faces are compatible. The process of measuring face-level relational evidence for a face's label is shown in Fig. 6.9. Panels (a) and (b) of this figure show the face-level elements on the model panel and the data panel for a simple scene, respectively. Assume for example that face F_3, whose label is F_B, is being used to update the belief in the label (F_C) of face F_5. Panel (c) of this figure shows how the transformation, $T_{F_C \rightarrow F_B}$, is defined to force the centroids of faces F_B and F_C coincide. Panel (d) of this figure shows the result of applying this transformation to face F_5. To check the (in)compatibility of the two faces, the (non)collocate metrics would be applied using the centroids of face F_3 and the transformed version of face F_5 (the centroids are shown here as the crosses inside the faces).

In reality, a single procedure is used for enforcing both the local and the relational constraints within a group. If two elements have the same label, then the identity transformation is used in the updating procedure. If the identity transformation, $T_{E_x \rightarrow E_x}$, is used to check relational constraints, then the (in)compatibility calculations reduce to the computations required for (in)compatibility calculations for mutual consistency.

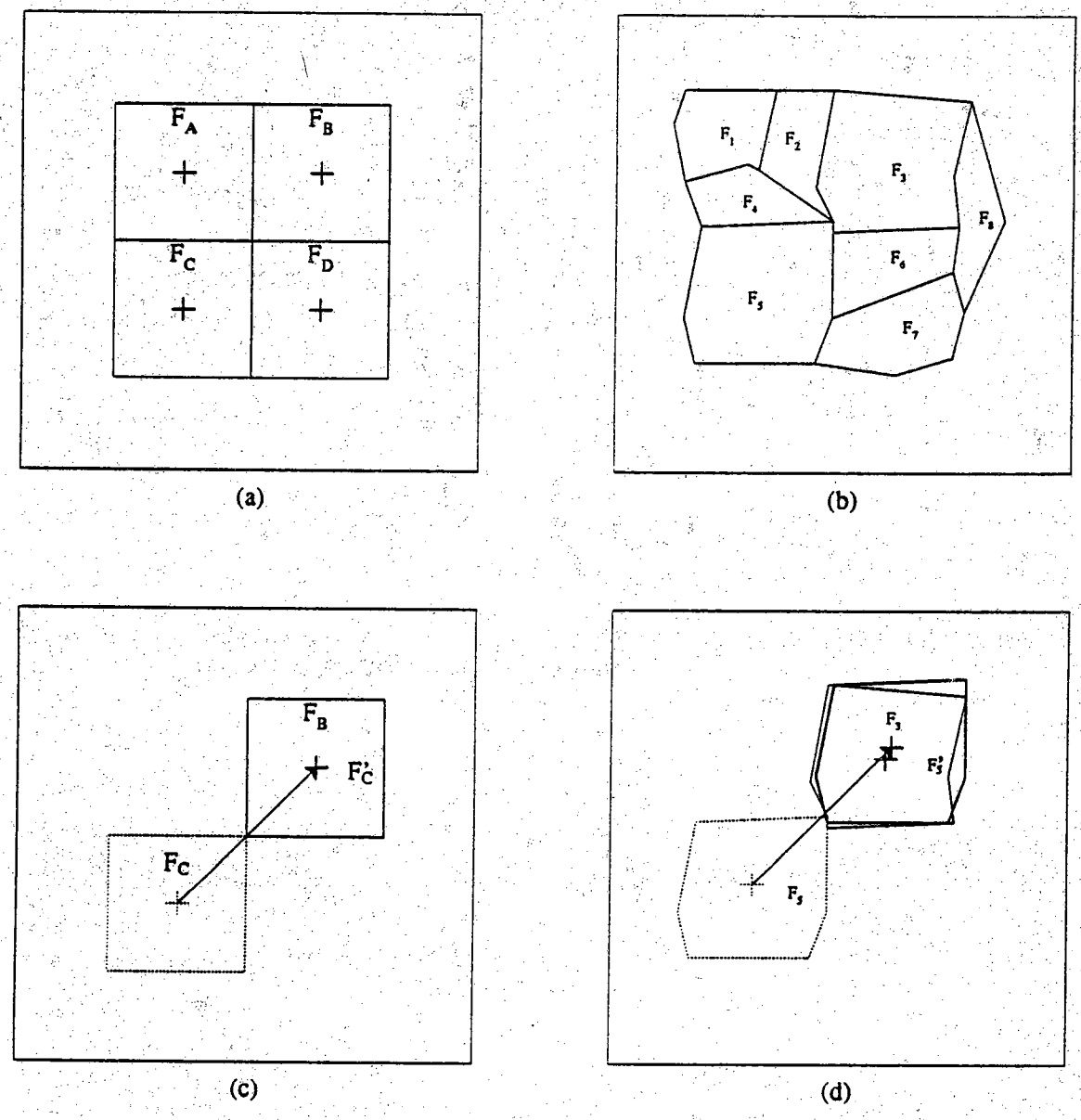


Figure 6.9 This figure shows the the processes used to check the relational constraints for face elements with different labels.

6.4. An Example of the Labeling Process

An example of the processes used to label faces and update the belief in the labels may help to clarify the concepts introduced in this chapter. In this example, assume that the expected scene consists of a single object with four faces, as shown in panel (a) of Fig. 6.9. Also assume that a region-based preprocessor presented PSEIKI with the observed scene depicted in panel (b) of Fig. 6.9. The first step in the labeling process consists of determining the FODs for the faces on the data panel. As previously described, a model element is include in an element's FOD if the data element's expanded extent overlaps the extent of the model element. For example, if F_A was the only model face whose extent overlapped face F_1's expanded extent, then F_1's FOD would consist entirely of $\Theta_{F_1} = \{F_A\}$. On the other hand, if the extent of face F_6 overlapped with the extents all of the model faces, then its FOD would be $\Theta_{F_6} = \{F_A, F_B, F_C, F_D\}$. Assume that the faces received the following FODs

$$\Theta_{F_1} = \{F_A\}$$

$$\Theta_{F_2} = \{F_A, F_B\}$$

$$\Theta_{F_3} = \{F_A, F_B\}$$

$$\Theta_{F_4} = \{F_A, F_B\}$$

$$\Theta_{F_5} = \{F_A, F_C\}$$

$$\Theta_{F_6} = \{F_A, F_B, F_C, F_D\}$$

$$\Theta_{F_7} = \{F_C, F_D\}$$

$$\Theta_{F_8} = \{F_B, F_D\}$$

After the FODs for the face elements have been determined, the initial belief function of each element is computed by measuring the *ES_compat()* and *ES_incompat()* between that face's convex hull and the convex hulls of the model elements in its FOD. For example, the following (in)compatibility measures could result from applying the initialization metrics between face F_6 and the model faces.

$$ES\_compat(F_A, F_6) = 0.05$$

$$ES\_incompat(F_A, F_6) = 0.75$$

$$ES\_compat(F_B, F_6) = 0.1$$

$$ES\_incompat(F_B, F_6) = 0.65$$

$$ES\_compat(F_C, F_6) = 0.1$$

$$ES\_incompat(F_C, F_6) = 0.72$$

$$ES\_compat(F_D, F_6) = 0.35$$

$$ES\_incompat(F_D, F_6) = 0.25$$

The following SEFs are obtained for face F_6 by applying the process described in appendix B.

$$M^{F_A}(F_A) = 0.05 \qquad M^{F_B}(F_B) = 0.1$$
$$M^{F_A}(\neg F_A) = 0.75 \qquad M^{F_B}(\neg F_B) = 0.65$$
$$M^{F_A}(\Theta) = 0.20 \qquad M^{F_B}(\Theta) = 0.25$$

$$M^{F_C}(F_C) = 0.1 \qquad M^{F_D}(F_D) = 0.35$$
$$M^{F_C}(\neg F_C) = 0.72 \qquad M^{F_D}(\neg F_D) = 0.25$$
$$M^{F_C}(\Theta) = 0.18 \qquad M^{F_D}(\Theta) = 0.40$$

With these initial SEFs, the belief in face F_6's singletons can be computed using the procedure described in chapter 5.

$$Bel(F_A) = 0.0607$$

$$Bel(F_B) = 0.1090$$

$$Bel(F_C) = 0.0946$$

$$Bel(F_D) = 0.6880$$

Thus face F_6 would be assigned the initial label F_D with belief 0.69. The same process is used to initialize the belief functions of the other face elements on the data panel. Assume that the faces received the labels shown in table 6.1.

Table 6.1 This table shows the initial labels assigned to the faces in the example.

Face	Label	Belief
F_1	F_A	0.30
F_2	F_A	0.42
F_3	F_B	0.72
F_4	F_A	0.33
F_5	F_C	0.67
F_6	F_D	0.69
F_7	F_D	0.31
F_8	F_B	0.20

After each face's belief function has been initialized, the grouper KS is allowed to group compatible faces into objects. If one of the groups formed by the grouper KS consists of faces F_1, ..., F_7, then these faces can be used to update the belief in each other's labels. For example, to update the belief in face F_6's label, its (in)compatibility with each of its siblings is checked using the *collocate()* and *noncollocate()* metrics and the appropriate transformations. Because F_6 and F_7 have the same label, the updating evidence provided by measuring their consistency is computed as follows (assuming that $SF_{face-updating}$ is equal to 1.0)

$$\text{collocate}(F_7, F_6) = 0.4$$

$$\text{noncollocate}(F_7, F_6) = 0.5$$

These measures are then converted into an updating SEF.

$$M^{F_D}_{F_7 \to F_6}(F_D) = Bel_{F_7}(F_D) \times \text{collocate}(F_7, F_6) \times SF_{face-updating}$$

$$= 0.31 \times 0.4 \times 1.0$$

$$= 0.124$$

$$M^{F_D}_{F_7 \to F_6}(\neg F_D) = m_{F_7}(F_D) \times \text{noncollocate}(F_7, F_6) \times SF_{\text{face-updating}}$$
$$= 0.31 \times 0.5 \times 1.0$$
$$= 0.155$$

$$M^{F_D}_{F_7 \to F_6}(\Theta_{F_6}) = 1.0 - 0.13 - 0.18$$
$$= 0.721$$

However, since the other faces in the group do not have the label F_D, face F_6 must be transformed before the metrics are applied. For example, because face F_5 and face F_6 have different labels, the the transformation $T_{F_D \to F_C}$ must be used in the (in)compatibility measurements.

$$\text{collocate}(F_5, T_{F_D \to F_C}(F_6)) = 0.8$$
$$\text{noncollocate}(F_5, T_{F_D \to F_C}(F_6)) = 0.2$$

These values can then be used to form the updating SEF.

$$M^{F_D}_{F_5 \to F_6}(F_D) = Bel_{F_5}(F_C) \times \text{collocate}(F_5, T_{F_D \to F_C}(F_6)) \times SF_{\text{face-updating}}$$
$$= 0.67 \times 0.8 \times 1.0$$
$$= 0.53$$

$$M^{F_D}_{F_5 \to F_6}(\neg F_D) = Bel_{F_5}(F_C) \times \text{noncollocate}(F_5, T_{F_D \to F_C}(F_6)) \times SF_{\text{face-updating}}$$
$$= 0.67 \times 0.2 \times 1.0$$
$$= 0.14$$

$$M^{F_D}_{F_5 \to F_6}(\Theta_{F_6}) = 1.0 - 0.53 - 0.14$$
$$= 0.33$$

Updating evidence can be generated by checking face F_6's consistency with the other faces in the group in a similar manner. After the all of the faces in the group have been used to provide evidence on the validity of face F_6's label, the resulting updating SEF is combined with the corresponding SEF in F_6's belief function to yield a new belief function.

Note that, in this example, the effects other KSs have on the processing have not been discussed. For example, the merger KS most likely would merge the following groups of faces at some point in the processing because the elements in each group are adjacent and have the same label.

$$\{F_1, F_2, F_4\} \rightarrow F_9$$

$$\{F_6, F_7\} \rightarrow F_{10}$$

The composite faces formed by the merger then would be labeled and updated in the manner described above. In the next chapter, the methods used by the splitter, merger and grouper KSs to create and modify groups will be described.

6.5. Independence Considerations of the Evidence Metrics

The following question is frequently raised regarding evidence accumulation in PSEIKI: Is the necessary condition for the application of Dempster's rule satisfied? This condition states that all evidence must come from disparate sources, i.e., the sources of evidence must be independent. In this section, we will explore the independence of evidence generated using the metrics introduced in this chapter.

To start the argument for the independence of the evidence sources, we should note that the term "independence" does not mean statistical independence when used in the context of the Dempster-Shafer theory. Rather, the term takes on a more philosophical connotation. In the D-S framework, the term "independence" means the lack of predictability. That is, can the evidence contained in one belief function be used to predict the distribution of evidence contained in another? If not, then the two belief functions are said to be independent. Using this definition, we will argue that all of PSEIKI's SEFs combined into an element's final belief function are independent. In these arguments, we will focus on establishing the independence of the masses focused on the singleton propositions in each SEF (i.e. $M^{\theta_i}(\theta_i)$). Because, in each SEF, the masses of the other focus elements depend on this value, the belief values contained in entire SEF are independent if the singleton masses are independent. As will be seen, most of the arguments establishing SEF independence hinge on the observation that the elements from PSEIKI's observed scenes fall into two categories. First, there are elements appearing randomly in the observed scene that do not correspond to any element in the expected scene. These elements are often caused by noise in the image, unmodeled shading affects and

peculiarities of the low-level vision system. Because of their completely random nature, we do not need to worry about the independence of the evidence provided for these elements' labels. Second, there are elements in the observed scene that correspond to elements from the expected scene. However, these observed elements do not appear exactly as they were predicted to appear in the expected scene. Instead, they are randomly perturbed from their expected appearance because of noise in the images, segmentation affects, etc. Thus, one is not able to predict how the parameters of one element (such as position, orientation or shape) has been perturbed given the parameters of another model or data element. To guarantee that all the evidence accumulated into a data element's final belief function is from disparate sources, the following four pairs of SEF types must be shown to be independent†.

1) First, the SEFs that are combined to form the element's initial belief function must be independent. That is, $M^{\theta_i}(\cdot)$ must be independent of $M^{\theta_j}(\cdot)$ for all $i \neq j$. This independence requirement can be met only if it is not possible to predict the value that an expected-scene metric produces when applied to one model edge given its value when applied to another model edge. On the edge level, this requirement stipulates that for any data edge, E_i, the value produced by the compatibility metric when applied to one model edge, ES_edge_compat(θ_A, E_i), should not be predictable given the metric's value when applied to any other model edge, ES_edge_compat(θ_B, E_i), where θ_A and θ_B are model edges from its FOD. The random nature of the data elements' perturbations cause the initial evidence values for this case to be independent. Because the initialization metrics are composed of number of parts (e.g. the face-level metrics are based on shape and distance constraints), it is not possible to tell what portion of the (in)compatibility values are due to which part of the metric. Because it is not possible to attribute the (in)compatibility values to specific parts of the metrics, it is not possible to predict the values produced by the metrics with one model element given the values produced by the metrics with another model element.

2) Second, the updating SEFs must be independent of the initialization SEFs. Formally, given a data element ψ_i, all of its initialization SEFs, $M^{\theta_i}_{\psi_i}(\cdot)$, must be independent of all its updating SEFs, $M^{label}_{\psi_j \rightarrow \psi_i}(\cdot)$. Superficially it may seem that

† The arguments given here address only edge-level independence considerations, equivalent face-level and object-level arguments exactly parallel those given here.

PSEIKI violates this condition because the updating evidence is based on initial belief functions. Remember, the updating evidence is the product of the compatibility value and the belief in the updating element's label. However, it is impossible to predict the updating evidence based on the initialization evidence for the following two reasons. First, the belief functions on which the updating SEFs are based are associated with the element's siblings -- not the element in question. The initial belief functions of an element's siblings are independent of the element's initial belief function for the reasons described in case (1). Second, the updating belief functions are formed by multiplying the belief in the element's siblings' labels (an unpredictable value) by the output of the compatibility metric (a predictable value). Because the result of multiplying a predictable value by an unpredictable value is unpredictable, the evidence is independent. Hence, the independence requirements for the application of Dempster's rule are not violated.

3) Third, the updating SEFs from all of the element's siblings must be mutually independent. That is, given any two elements, ψ_j and ψ_k providing updating evidence for another element, ψ_i, $M^{label}_{\psi_j \rightarrow \psi_i}(\cdot)$ must be independent of $M^{label}_{\psi_k \rightarrow \psi_i}(\cdot)$. The argument for this case follows exactly from the argument for case (2). Once again, the initial belief functions for the element's siblings are mutually independent. Thus, the updating evidence provided by one sibling cannot be predicted by the updating evidence provided by another.

4) Fourth, if a data element has its label changed, then the updating SEFs focused on the new label must be independent of the updating SEFs focused on its old label. Formally, without loss of generality, assume that the first two labels of a data element, ψ_i, are *label(1)* and *label(2)*, respectively. Then if ψ_j is used to update the belief in ψ_i's label, the evidence that it provides focused on the first label, $M^{label(1)}_{\psi_j \rightarrow \psi_i}(\cdot)$ must be independent of the evidence that it provides focused on the second label, $M^{label(2)}_{\psi_j \rightarrow \psi_i}(\cdot)$. The argument for this case follows the argument from case (1) exactly. Because the observed elements correspond to expected scene elements but have been randomly perturbed, the relationship (an hence the metric values) between a data element and one model element is unpredictable given the metric values between the data element and another model element.

An experiment was undertaken to lend additional strength to the assertion that the evidence used in PSEIKI is independent. In this experiment, the evidence produced by the edge-level metrics was investigated for each of the four cases described above. The sample correlation coefficient, $\hat{\rho}$, over 10,000 trials was chosen as the measure of independence; the evidence would be deemed to be independent only if $\hat{\rho}$ was small for each of the four cases listed above.

Two model edges and three data edges were used in this experiment. All of these edges were restricted to lie on the *xy-plane*. The two model edges, E_A and E_B, were 100 units long and lay along the x and y axis, respectively. The three model edges, E_1, E_2 and E_3, also were 100 units long. In each trial, each data edge was randomly assigned a "true" model edge; the edges were considered to correspond with the true models, but their position was slightly perturbed (the "correct" edge assignment and perturbations were different for each trial). For example, the center of the data edge was randomly offset from the center of the model edge; the offsets were uniformly distributed such that the middle of the data edges fell in a disc with radius of 10 units centered on middle of the model edge. The data edges also were rotated about their centers by a random amount; the rotations were uniformly distributed from $-\pi/16$ to $\pi/16$. After the data edges' geometry was initialized, the edge-level metrics described earlier in this chapter were used to generate the evidence about edge E_1's label. The maximum allowed misregistration used by the metrics, D_{max}, was set to 25. The edges were treated as if they were grouped into a face; thus, they were allowed to provide updating evidence for each other. The evidence accumulated into edge E_1's SEFs were stored for each trial and were used to determine $\hat{\rho}$ at the end of all 10,000 trials.

After all trials were complete, the sample correlation coefficient was determined for each of the four cases described above using the values stored from each of the 10,000 trials. In the first case, $\hat{\rho}$ was determined for edge E_1's initial probability masses, $M^{E_A}(E_A)$ and $M^{E_B}(E_B)$. The following equation was used to compute $\hat{\rho}$.

$$\hat{\rho}(X, Y) = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_i (x_i - \bar{x})^2 \sum_j (y_j - \bar{y})^2}}$$

where $\bar{x}$ and $\bar{y}$ were the sample mean values of the masses. Using this equation, the magnitude of $\hat{\rho}$ was determined to be less than 10^{-13}.

The sample correlation coefficient for the second case was determined using edge E_1's initialization and updating SEFs. If E_1's initial label was determined to be E_A, then $\hat{\rho}$ was determined from the pair $M^{E_A}(E_A)$ and $M^{E_A}_{E_2 \to E_1}(E_A)$. However, if E_1 assumed E_B as its initial label, then $\hat{\rho}$ was determined from the pair $M^{E_B}(E_B)$ and $M^{E_B}_{E_2 \to E_1}(E_B)$. In this case, the magnitude of $\hat{\rho}$ was determined to be less than 10^{-15}.

The sample correlation coefficient for the third case was determined using the updating evidence focused on edge E_1's label. If E_1's initial label was determined to be E_A, then $\hat{\rho}$ was determined from the pair $M^{E_A}_{E_2 \to E_1}(E_A)$ and $M^{E_A}_{E_3 \to E_1}(E_A)$. However, if E_1 assumed E_B as its initial label, then $\hat{\rho}$ was determined from the pair $M^{E_B}_{E_2 \to E_1}(E_B)$ and $M^{E_B}_{E_3 \to E_1}(E_B)$. In this case, the magnitude of $\hat{\rho}$ was determined to be less than 10^{-14}.

The sample correlation coefficient for the fourth case was determined using the trials in which disconfirmatory evidence focused on edge E_1's initial label forced the label to change. Initially, there were not enough trials in which edge E_1's label was changed to reliably determine $\hat{\rho}$. However, when the amount the data edges were allowed to deviate from their true model was increased (by increasing the amount that they were offset and rotated), then $\hat{\rho}$ was determined for the pair $M^{E_A}_{E_2 \to E_1}(E_A)$ and $M^{E_B}_{E_2 \to E_1}(E_B)$. As in the first three cases, the magnitude of $\hat{\rho}$ was very small; in this case, $\hat{\rho}$ was determined to be less than 10^{-11}. These sample correlation coefficients are small enough to confidently call the evidence uncorrelated. Given these small sample correlation coefficients, and the above arguments, we are confident that our use of metrics to generate evidence meets the independence prerequisites for the application of Dempster's rule.

CHAPTER 7

BLACKBOARD OPERATION

PSEIKI's architecture is shown in Fig. 7.1. The blackboard (BB) data structure is split into two panels. The data panel holds the abstraction hierarchy for the observed scene and the model panel holds the abstraction hierarchy for the expected scene. The knowledge PSEIKI uses to accomplish its task is partitioned into its four knowledge sources (KSs), the grouper KS, the merger KS, the splitter KS and the labeler KS. The remaining two components, the monitor and scheduler, are used to determine which KS should be invoked on each BB processing cycle.

PSEIKI's image preprocessors produce data only for the lower levels of the blackboard; thus, the system needs to generate data elements on higher levels. It is the grouper KS's task to create data elements on the upper levels of the hierarchy by forming groups of lower-level elements that meet geometric constraints exhibited by the model data. Data presented to PSEIKI by its low-level preprocessors often is far from optimal. Many times image structures that should remain separate are merged into a single structure (i.e. the image is undersegmented) or an image structure is incorrectly broken into a number of smaller ones (i.e. the image is oversegmented). In fact, it is common for a single image to be undersegmented in one section and oversegmented in another. The splitter and merger KSs are designed to correct grouping errors produced by the low-level preprocessors and the grouper KS. The merger KS tries to correct oversegmented images by merging elements on one level of the blackboard into a single element on the same level. The splitter KS's task is to break an element into smaller elements, all of which reside on the same level of the blackboard as the original element. This splitting is done to correct for misformed elements created by the grouper KS. These two KSs use many of the classic splitting and merging techniques described in [BriFen70], [HorPav74],

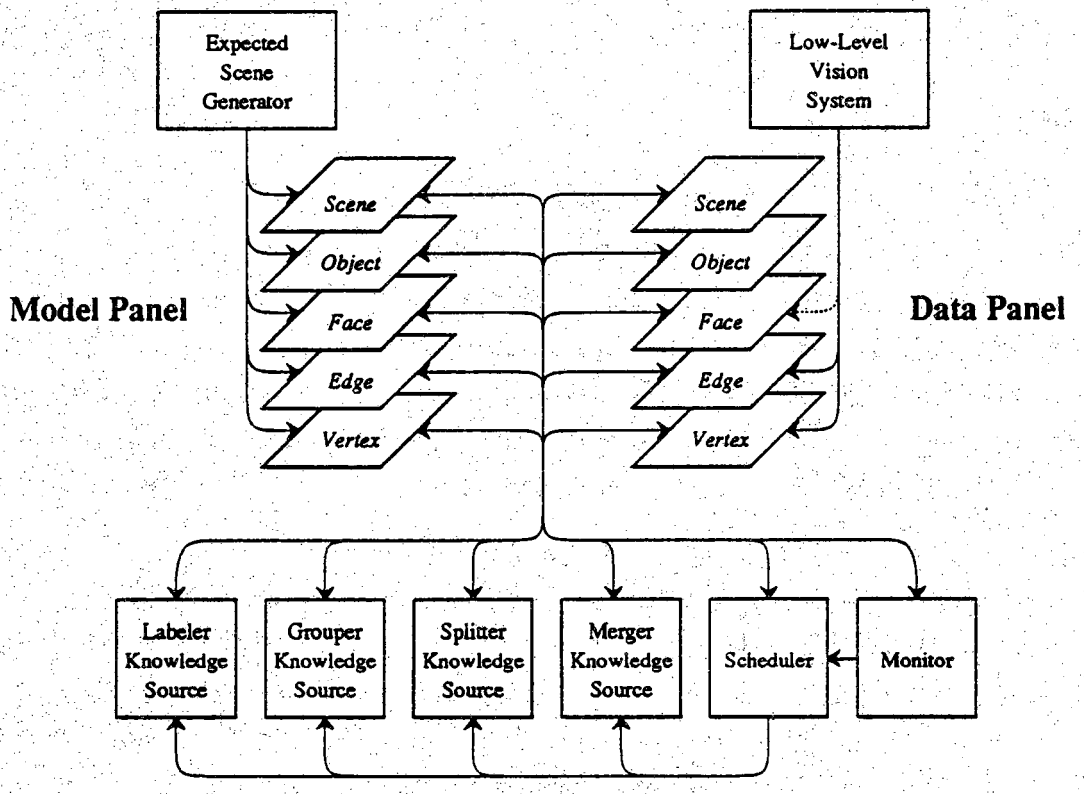


Figure 7.1 This figure shows the current configuration of PSEIKI's architecture.

[Zuc76], and particularly those expressed as rules in [NazLev84].

At the start of BB processing, PSEIKI queries the user to supply needed information. This information includes the name of the input files containing the image data, expected-scene information and camera calibration matrix. PSEIKI also queries for the maximum amount of scene misregistration and the desired number of competing data elements for each model element. PSEIKI then reads the input files and backprojects the image data onto the ground ($z = 0$) plane if appropriate. The user can request that the image data be backprojected onto the ground plane if the scene is expected to be essentially two dimensional (e.g. the scenes observed by a mobile robot following a sidewalk will be 2-dimensional). Otherwise, PSEIKI's data is operated on in the image plane. Other data needed by the KSs also are initialized at this time. For example, the convex hulls for all model elements on the face level and higher levels are initialized. The homogeneous transformation matrices needed by the labeler to update the belief in the labels of data elements also are initialized. After all the data on the BB has been initialized, PSEIKI's scheduler is invoked to start BB processing.

PSEIKI's results are output when the blackboard processing is complete. The scene level element with the largest belief is chosen as the final scene interpretation and its descendents are output using the format described in section 3.1. PSEIKI also reports statistics about its processing; for example, it reports the original number of elements deposited on each level by the preprocessor and expected scene generator, number of KS invocations, number of rules fired, number of OPS83 conflict set resolution tests, etc.

7.1. Monitor and Scheduler Operation

Together, a blackboard's scheduler and monitor have the task of choosing among the triggered KSs during each cycle of processing. The data structure used to hold a KS's triggering information is called a Knowledge Source Activation Record (KSAR). A KSAR is created by the BB monitor when the preconditions for a KS are satisfied by some data-elements on the BB. At the start of each BB cycle, the scheduler chooses among KSARs to determine which KS to fire. In PSEIKI, KSARs also can be created by KSs, allowing KSs to trigger other KSs explicitly. For example, the grouper KS triggers the labeler KS by building a KSAR whenever it forms a new element from a group of lower-level elements; it builds this KSAR because it is known that the new element will need to be labeled. Allowing KSs to trigger each other is a form of direct KS

communication, and it reduces their independence somewhat. However, the efficiency gained by allowing this highly structured form of KS communication justifies the slight loss of KS independence.

7.1.1. Monitor Operation

The monitor is the watchdog of the BB. It is the monitor's job to keep track of the data on the BB and trigger the KSs when their preconditions are met. When the monitor determines that a KS's trigger conditions have been met, it builds a KSAR. Each KSAR holds the identity of the relevant KS, the triggering data-element, and other pertinent information such as the cycle during which the KSAR was created. This information indicates the object on which work should be performed and aids the scheduler in choosing which KSAR to activate.

A status flag is associated with each KSAR. Initially, the monitor creates each KSAR with the status flag set to *pending*. This value of the the status flag indicates that the KS has been triggered but has not been run. When the scheduler uses a KSAR to fire a KS, it sets the KSAR's status to *active*. When a KSAR's status is set to active, the monitor is invoked to guarantee that the KS's preconditions have not become invalid since the KS was triggered. If the monitor determines that the KSAR is invalid, it sets the status flag to *poisoned* and returns control to the scheduler. If the KS's preconditions are still valid, then the monitor sets the status flag to *running* and passes control to the KS. When the KS is finished running, the KSAR's status is set to *finished*; either the KS or the monitor can set the flag to this value. The monitor also is responsible for watching the blackboard to determine if the status of any poisoned KSARs should be reset to pending. The KSAR's status is reset if the KS's preconditions are once again met by the specified data-element.

7.1.2. Scheduler Operation

The scheduler is the heart of any BB. It is the scheduler's task to choose what action to perform during each cycle of the BB's operation. It carries out this job by selecting one of the pending KSARs and activating the corresponding KS.

PSEIKI's scheduling strategy can be broken into three phases. The first phase is called the *initialization* phase. In this phase, the labeler KS assigns labels to the elements deposited on the data panel by the low-level processor; the grouper KS and labeler KS also are used to create and assign labels, respectively, to elements on the upper levels of the data panel. In the second phase, called the *updating* phase, the belief in the labels of the data elements is updated. The third phase is called the *propagation* phase; in this phase, evidence is passed up the hierarchy and is incorporated into the upper-level elements' belief functions.

Although the scheduling algorithm generally follows this three phase pattern, the actions usually designated to one phase may be performed in another phase if the need arises. For example, the labeler KS must be called to initialize the label of any newly created data element, regardless of the scheduling phase. For example, if in the middle of the updating phase the labels of two adjacent faces become identical, then the merger KS will merge the two faces into a single grouping. After the merger creates this new element, the labeler KS will be invoked to initialize its label. Also, if an upper-level element's label is changed during the updating or propagation scheduling phases, then the element's descendents are assigned new labels via the three-phase process. For example, if a face receives a large amount of disconfirmatory evidence forcing its label to change, then the belief functions of all of its component edges will be cleared and reinitialized, regardless of the processing phase.

The scheduling of the the merger KS also falls outside of the initialize/update/propagate process described above. Scheduling the firing of the the merger is viewed as an exceptional event which is not part of the normal KSAR selection process. Because this KS has a higher priority† than the grouper and the labeler, the scheduler will fire this KS as soon as one of its KSARs appears, regardless of the current scheduling phase. It seems reasonable to fire this KS first because its duty is to correct misformed elements; any processing resources spent labeling such an element or including such an element in a group most likely will be wasted. Thus, it makes sense that these misformed elements be corrected as soon as possible. Because the scheduling of the merger is an exceptional event, PSEIKI's scheduling scheme can best be described as the initialize/update/propagate process described above with opportunistic interruptions; the interruptions occur whenever a pending merger KSAR appears or when the labeler

† The priority of all of the KSs are programmed into the system before runtime.

KS changes an element's label. When one or more priority KSARs appears in the working memory, the scheduler ranks them based on their priority and chooses the highest ranked KSAR for firing. If two or more KSARs have the same maximum priority value, then one is selected at random. The scheduler will continue to fire these KSARs until there are no pending priority KSARs with a priority greater than a predefined threshold left in working memory.

Two actions are performed during the initialization phase of BB processing. First, data elements on the lower levels of the BB are grouped into upper-level elements by the grouper KS. Second, the labels and belief functions of unlabeled data elements are initialized by the labeler KS. The reasoning performed by the scheduler during the initialization phase is primarily goal-driven. Scheduling is started with the goal of finding a prespecified number of instantiations† of the scene-level model element. Each model element has a desired number of instantiations specified by the user at the start of BB processing, n_G. However, the number can be changed by the scheduler during processing if needed. For example, the number for a particular model element can be decreased if the desired number of elements with the proper label can be created.

Initially, the scheduler is unable to meet its goal of finding the scene-level instantiations because only the lowest two or three levels of the data panel contain elements. Thus, the scheduler must form sub-goals to enable it to meet its final goal. If the model scene's object-level children each had a sufficient number of data element instantiations, then the scheduler could fire the grouper KS to group them together into a scene-level data element that could be matched with the model element. Therefore, the scheduler forms a sub-goal to find a prespecified number of data element instantiations for each of the model scene's child-objects. Once the instantiations of its children are found, the scheduler fires the grouper KS to group them together and the labeler KS to label the newly formed element.

Sub-goals are created to find n_G instantiation data elements on successively lower levels of the BB until a level containing data elements is reached. If an edge-based preprocessor is used to generate PSEIKI's input data, then the edge level is the highest level with data elements on it; if a region-based preprocessor is used, then the face level is the highest level with data elements on it. When a sub-goal is created to find an

† Data elements that have been matched with a model element are called *instantiations* of that model element.

element on a non-empty level, all data elements on that level are labeled. Note that only the highest level elements deposited onto the data panel are labeled at this time (e.g. faces for a region-based preprocessor, edges for an edge-based preprocessor); the labels for elements on the lower levels are not initialized until the updating phase of BB processing.

After all of the labels for these elements have been assigned, the scheduler tries to satisfy its lowest level sub-goals by creating model instantiations one level up in the hierarchy. To find a data element that can become a model element's instantiation, the scheduler fires the grouper KS to form a new element on the correct level (later, the labeler KS will be fired to give the element an initial label). The scheduler picks the grouper KSAR whose focus element is an instantiation of one of the model element's children and whose label has the largest belief; the focus element will become the seed for the higher-level element created by the grouper KS. After the grouper KS forms an element, the splitter KS is triggered to check if any competing elements were included in the group. The scheduler fires on this KSAR immediately (splitter KSARs have a high priority) and the splitter splits the group if necessary. The scheduler then fires the labeler KS to find the initial label of the new element (or elements if the splitter found competing children). Note that the labeler KS is free to assign any label to the newly created element even though the grouper KS was fired to create a data element that would become an instantiation for a particular model element. For example, assume that the scheduler has the goal of creating an instantiation for a model face, say face F_A. To satisfy the goal, the scheduler would fire the grouper to form a new face-level data element and then the labeler to assign it a label. Thus, even though the grouper was fired to form a new face to be labeled F_A, the labeler is not required to assign the label F_A to that new face. While in practice such a transfer of labels is not very likely, the labeler is given this freedom for the sake of a homogeneous computational procedure.

A sub-goal is satisfied when the n_G instantiations of a model element are created. If it is not possible for the KSs to create enough instantiations for a model element, then n_G is decreased until the goal is satisfied[†]. This process is akin to using a depth bound for finding a solution in a search graph. When all of a goal's sub-goals are satisfied (i.e. all of the model's children have the correct number of instantiations) the scheduler tries to

[†] In the future, other actions may be used to allow the KSs to create n_G instantiations. For example, the constraints used to determine if elements should be grouped together could be adjusted such that new groups are formed.

satisfy the higher-level goal by applying the group/split/label process to the newly created elements. If any new elements are created by the merger during this phase of BB processing, the labeler is called to initialize its label.

It should be clear that in the initialization phase, the operation of the scheduler combines top-down model-driven search with bottom-up data-driven requests created by the monitor. Combinatorial explosions are controlled by putting an upper bound on the number of competing hypotheses that can be entertained in the model-driven search. It is important to note that the number of competing hypotheses for any model node is not limited to n_G. To explain, assume that the there are $n_G - 1$ instantiations of a model element on the data panel. To meet the goal, the grouper KS would then be fired to form a new data element. Since the splitter KS is given a high priority by the scheduler, this KS will fire next and may discover that a number of the new element's children compete. Therefore, the splitter KS will split the group into smaller groups, each with one of the competing children. In other words, because of the action of the splitter KS, there can be a geometrical multiplication of the hypotheses formed by the grouper KS. For these reasons, it becomes necessary to give a small value to n_G; currently, n_G is usually set to 3.

After the initialization phase of processing is completed, the updating phase is started. Backward chaining is used extensively during this phase of BB processing. First, the labels of all of the children (All of which will reside on the object-level) of the in-focus scene-level elements are updated. Next, the labels of all of the children of the object-level children are updated (All of these elements will reside on the face-level). Note that all of an element's children are updated simultaneously to prevent an element for lending evidence to itself indirectly (by lending evidence to one of its siblings which, in turn, would lend evidence to the original element).

This updating process proceeds down the data-panel hierarchy in a depth-first manner until the edge-level is reached. If an edge-based preprocessor was used to provide the input data, then labels will have been assigned to the edges during the initialization phase; in this case, the edges' belief functions are updated normally. However, if a region-based preprocessor was used to provide the input data, then the edges on the data panel will not have been labeled during the initialization phase. If the scheduler tries to update the belief in the labels of an element's children which are without initial labels, then the children's labels are initialized first. Delaying the assignment of the edges' labels until after their parent faces' labels have been updated eliminates the need to change the edges' FOD if their parents' labels are changed.

If an element's updating evidence causes its label to change during this phase of BB processing, then the scheduler fires the labeler KS to clear all of the label information (including FODs, labels and belief functions) of the element's descendents. Next, the labeler KS is scheduled to update the belief in the element's new label. Finally, a goal is generated to update the belief in the element's descendents' labels. Because all of the descendents' label information was just purged, the scheduler fires the labeler KS to initialize the descendents' information before firing the labeler KS to update the labels. If the splitter KS or merger KS create a new element during this phase, then the element's label is initialized and updated immediatedly.

During the propagation phase of BB processing, the updating evidence generated by checking the consistency of the element's descendents' labels is accumulated into an element's belief function. In this phase, the evidence generated during the updating phase of processing is passed up the hierarchy from the edges into their parent faces then from the faces to their parent objects, etc. Because the monitor forms propagation KSARs only for scene-level data elements, the scheduler randomly picks propagation KSARs to fire until there are no pending KSARs left. If an element's descendents' labels were inconsistent enough to cause the element's label to change, then the initialization/update/propagation process just described immediately is applied to the element's descendents. Likewise, if the splitter or merger create a new element during this phase, then the initialization/update/propagation process immediately is applied to the new element. The propagation scheduling phase is complete when there are no pending propagation KSARs and no pending splitter or merger KSARs. When this phase of BB processing is complete, PSEIKI outputs its processing results.

7.2. Knowledge Source Operation

7.2.1. Operation of the Grouper Knowledge Source

The grouper KS builds data elements on the upper levels of the hierarchy from data elements deposited by the low-level vision system. The grouper KS builds the upper level elements in a data-driven manner by grouping objects on the lower levels of the hierarchy into progressively higher levels. For example, if an edge-based preprocessor is used to generate input data, the grouper first groups edge-elements into faces and then

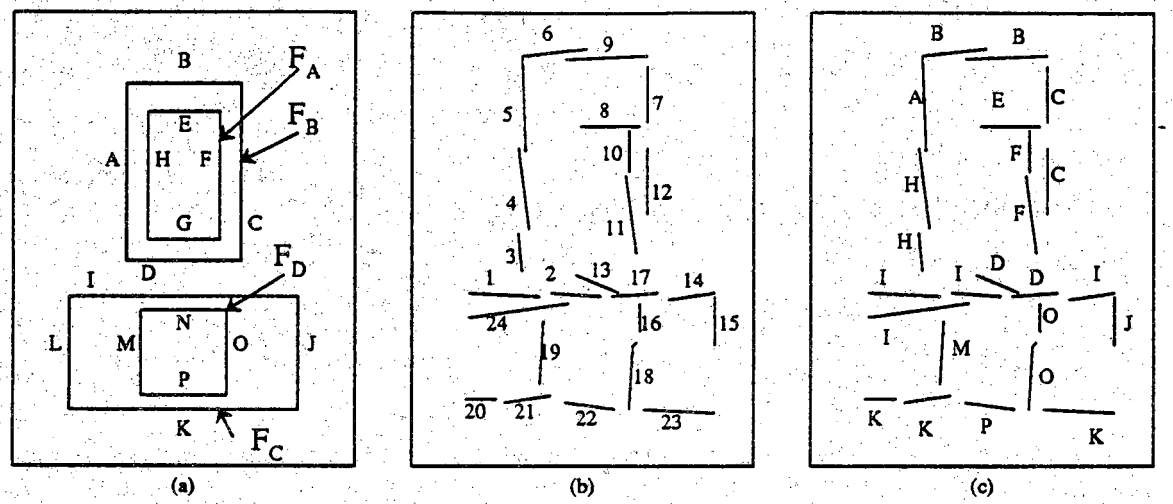


Figure 7.2 An expected scene is shown in panel (a), the edges produced by an edge-based preprocessor in panel (b), and their initial labels in panel (c).

groups the faces into objects, and so on.

Fig. 7.2 shows a simple example of how the grouping is performed; panel (a) shows the expected scene, panel (b) shows the edges presented to PSEIKI by an edge-based preprocessor, and panel (c) shows the initial labels for those edges.†

The grouper KS is triggered by the monitor when the monitor detects an orphan data-element that has been assigned a label. These orphan elements can have a number of origins: The low-level preprocessor deposits a large number of orphan elements onto the data panel at the beginning of processing; in fact, all edge-level elements deposited by an edge-based preprocessor are orphans, as are all face-level elements deposited by a region-based preprocessor. Any data element created by the grouper KS is an orphan initially, as are elements created from other orphan elements by the splitter KS or merger KS. When the KS is triggered by the monitor, a KSAR is built indicating that the orphan element should be used as a seed-element of a group. For example, all of the edges shown in Fig. 7.2(b) are assumed to be orphans deposited on the data panel by an edge-

† The labels shown in Fig. 7.2 are intended only for the purpose of explanation here. In actual practice, even for simple imagery, the initial labeling may be much more chaotic, depending upon the extent to which an image is degraded by noise and other artifacts.

based preprocessor at the start of processing; thus, a KSAR is built for each edge indicating that it should be grouped. Because the monitor builds at most one KSAR for each data element, an element can be the seed only for a single group. This restriction is currently needed to enable the monitor to efficiently watch the BB data. Future versions of PSEIKI may allow an element to be the seed for more than one group if the seed element is thought to be robust. After the monitor triggers the KS by building the KSAR, it is up to the scheduler to determine when the KS will fire and form the specified group.

The scheduler fires the grouper KS when an instantiation for a particular model element is needed. At this point, the scheduler determines all of the grouper KSARs whose seed-elements can be children of an element with the desired label and ranks them based on the elements' belief. For example, at some point in the course of blackboard processing, a new data element with label F_D may be desired (see Fig. 7.2). To form an element with this label, the scheduler would rank the grouper KSARs for edge elements with labels E_M, E_N, E_O and E_P, because these are the only elements that could be the children of a face with label F_D. The scheduler then chooses the highest ranked KSAR and fires the grouper KS. When the grouper KS is fired, it determines the set of all edges that could possibly become the the seed-element's siblings. This candidate set is determined by the edges' labels. In the example, assume that edge E_{19} was chosen as the seed-element. Then the only edges that could possibly become its siblings are edges E_{16}, E_{18} and E_{22} because these are the only edges whose model elements are children of face F_D. After the set of candidate siblings has been determined, the compatibility metric discussed in chapter 6 is used to determine which candidates get grouped with the seed element. A candidate element will be grouped with the seed-element only if the compatibility metric yields a value above a user-specified threshold. For example, if the compatibility threshold has been set to 0.5 and the following compatibility measurements were made

$$\text{collinearity}(E_{16}, T_{E_M \to E_O}(E_{19})) = 0.65$$

$$\text{collinearity}(E_{18}, T_{E_M \to E_O}(E_{19})) = 0.55$$

$$\text{collinearity}(E_{22}, T_{E_M \to E_P}(E_{19})) = 0.43$$

then the grouper would assert that the following set of edges should be grouped.

$$F_1 = \{E_{16}, E_{18}, E_{19}\}$$

After the set of edges meeting the grouping constraint are determined, the grouper creates

a parent-element on the face level with the set of grouped edges as its children. The same process also can be used to find the following initial groups of edges

$$F_2 = \{E_3, E_4, E_8, E_{10}, E_{11}\}$$
$$F_3 = \{E_1, E_2, E_{14}, E_{15}, E_{20}, E_{21}, E_{23}, E_{24}\}$$
$$F_4 = \{E_5, E_6, E_7, E_9, E_{12}\}$$

Note that the blackboard monitor would trigger the grouper KS as soon as these face elements were created and labeled because each of them would be an orphan initially. Also note that the grouper KS may incorrectly group some edges into the face. For example, small edges generated by noise may accidentally be included in a group. Also, the grouper may incorrectly include competing elements into a group; two elements are said to compete if they cannot both be present in a consistently labeled scene interpretation. For example, in F_3, edges E_1 and E_{24} compete with each other. Obviously, the grouper KS should include only one of these competing edges in any group. It is the job of the splitter KS to remove the incorrectly grouped edges from a face. The splitter KS also has the duty to generate multiple faces from a face containing competing edges; the faces that the splitter generates retain only one competing edge. The actions performed by the splitter KS will be explained in greater detail later in this chapter.

The grouper KS groups faces into objects using a similar procedure; however, the grouper uses the *collocate()* metric introduced in the last chapter to determine if a candidate face should be grouped with the seed face. For example, assume that labeler KS assigned the labels and belief values show in table 7.1 to the faces created in the last example.

Table 7.1 This table shows the labels and belief values of the faces in the example.

Face	Label	Belief
F_1	F_D	0.40
F_2	F_A	0.42
F_3	F_C	0.72
F_4	F_B	0.53

If a data element on the object level with label O_A, which is composed of faces F_A and F_B, is desired at some point in the blackboard processing, then the scheduler would rank the appropriate KSARs based on their face's belief values. Then the scheduler would fire the grouper KS with the highest ranked KSAR. For this example, assume that the scheduler fired the grouper with F_4 as the seed-element. After the KS is fired, the grouping process proceeds as follows. First, the grouper determines a set of candidate sibling faces based on the their labels. In this example, face F_2 would be the only candidate face because it is the only face with one of the labels, F_A or F_B. If the compatibility threshold was set to 0.5 and the grouper measured the following compatibility measurement

$$\text{collocate}(F_2, T_{F_B \rightarrow F_A}(F_4)) = 0.69$$

then F_2 would be grouped with F_4 to create the following face

$$O_1 = \{F_2, F_4\}$$

Finally, the grouper creates an object level data-element with the set of faces that met the grouping constraint as its list of children.

7.2.2. Operation of the Labeler Knowledge Source

The labeler KS can perform five actions. First, it can determine a data-element's initial belief function and assign a label to the element. Second, it can update the belief in an element's label by checking the consistency of its label with the labels of its siblings. Third, it can be used to update the labels of all of the children of an element by checking their mutual consistency. Fourth, it can incorporate, into an element's belief function, updating belief generated by the element's descendants. Lastly, it can reset all of the label information of an element's descendents when the element's label is changed. The theories on which most of these actions are based have been discused in great detail in the previous two chapters. This section will describe how the labeler KS applies these ideas to accomplish its task.

The labeler is triggered to initialize a data-element's label when the monitor detects an unlabeled element on the data panel. It also is triggered when the element's label information is cleared because the label of one of its ancestors just changed. In both cases, the labeler is triggered because the element has no label information. When the labeler is fired to initialize an element's label, it determines the element's FOD and initial belief function using the process described in sections 6.1 and 6.2. It then assigns a

label to the element using the process described in section 5.4.

When the monitor detects that the element's label has changed, it triggers the labeler KS to update the belief in the element's label. When the labeler is triggered to update an element's label, each of the element's siblings is used to provide updating evidence about the element's new label using the concepts described in sections 5.4 and 6.3.

The labeler also can be triggered to update the belief in the labels of all of the children of an element. The labeler KS is triggered to update the belief in an element's children's labels when the monitor detects that the parent element just received its initial label. When the labeler is fired to update the labels of the children, it updates the label of each child sequentially using the processes described in sections 5.4 and 6.3. To limit the amount of updating evidence that is generated, not all elements have their belief function updated and not all elements are used to update the belief functions of their siblings. For example, the belief in an element's label is not updated if the belief is above a predetermined threshold, *MAX_BELIEF*. These strongly believed labels are not updated because it is unlikely that enough disconfirmatory evidence will be generated to force the label to change. Furthermore, an element is not used to update the belief in its siblings' labels if the element's belief is not higher than another predefined threshold, *MIN_BELIEF*. These weakly believed labels are not used because they will not produce strong enough updating evidence to justify the resources spent generating the updating evidence and accumulating it into the siblings' belief functions. Using these thresholds also prevents the belief in an element's label from saturating as discussed in appendix A.

The labeler is triggered to propagate the updating information from a scene-level data element's descendents up the hierarchy into the element's belief function after the element's children's labels have been updated. When the labeler is fired to propagate the descendents' information, the labeler propagates the updating evidence up the hierarchy using the techniques described in section 5.5.

The labeler also can clear the label information of all of an element's descendents when updating evidence causes the element's label to change. The labeler explicitly triggers itself to clear the label information when it changes the label of an element on the face level or higher. When the labeler is fired to clear the information, it chains down the element's descendent hierarchy clearing the following information for each descendent: the element's label, its FOD, its belief function and its updating belief function.

7.2.3. Operation of the Merger Knowledge Source

The merger KS also performs a grouping process; however, this process does not build elements on higher levels of the hierarchy from elements on lower levels, as does the grouper KS. Instead, it combines multiple elements on the blackboard into a single, larger element on the same level as the original elements. It combines elements if it is believed that they all can be represented by a single element on the model panel. This combination process is used to correct grouping errors produced by the low-level preprocessor. For example, the merger KS may group together a series of short edge segments into a longer segment, or a set of faces into a single larger face, if it is believed that the low-level preprocessor incorrectly fractured those data elements.

The monitor uses a special two-step process to trigger the merger KS. When it detects two elements with the same label, the monitor builds a merger KSAR with its status flag set to *partially_triggered*. It then changes the status flag to *triggered* if the two elements meet the merger's preconditions. This two-step triggering process allows the monitor to efficiently eliminate most data-element pairs from consideration before applying the costly triggering criteria. Furthermore, the monitor does not guarantee that the focus elements of a KSAR meet the all of the merging criteria when it triggers the merger KS. Thus, the merger first must determine if the elements under consideration really need to be merged. The criteria are not checked more rigorously by the monitor because it is more efficient for the merger to check that the criteria are met only for those elements on which it is fired. For example, it is not feasible for the monitor to check the collinearity of two edges before it builds a KSAR to merge them; thus, the merger KS needs to determine if two edges are sufficiently collinear before it merges them.

Because the merger is designed to correct artifacts produced by the image preprocessor, it is not allowed to operate on all levels of the BB. If an edge-based preprocessor is used to generate PSEIKI's input data, then the merger is applied only to edge elements. If a region-based preprocessor is used, then the merger merges both edges and faces.

Many edge-based processors produce artifacts that break edges into smaller line segments; the merger groups these small edges together into a single, larger edge. Thus, if an edge-based preprocessor is used to generate PSEIKI's input data, the merger KS tries to correct this error by joining line segments if they have the same label, are close together and are highly collinear. The merger KS also combines, into a single edge, highly collinear edges that are joined at a degree-two vertex and that have the same label.

The KS will merge two edges only if the the perpendicular distance from both of the vertices of the shorter edge is less than a predefined distance from the line defined by the longer edge. This distance threshold is usually set to a relatively small value to keep the KS from merging two edges that should remain separate. When two edges are merged, the KS deposits on the blackboard a new edge element with one vertex from each of the two old edges; these vertices are chosen to give the new edge maximal length. Placing strong constraints on the edges to be merged prevents the merger from merging two edges that should remain separate.

The edges deposited by many region-based preprocessors also may need to be merged. In this case, the merger is triggered to merge the edges in a parent face after the face's edges are assigned their initial labels. When the KS is fired to merge the parent's edges, each edge is checked to see if it should be merged with its neighbors. When two edges are merged, the merger deletes the vertex they have in common and creates a new edge between the two remaining vertices. The merger also will merge the newly created edge with its neighboring edges if possible.

If a region-based preprocessor is used to generate PSEIKI's input data, then data-elements on the face level also may be merged. The merger KS will combine two adjacent faces if they have same label; faces are said to be adjacent if they contain an common edge. When two faces are merged, the merger deposits on the blackboard a new face element whose list of children is the exclusive-or of the two old lists of child edges. That is, an edge is include in the new face's list of children if it is the child of exactly one of the old faces. Forming the new face's list of children in this manner prevents the edges that form the border between the two old faces from being included in the new face's list of children. Some of the merger KS's actions are shown in Fig 7.3.

The parameters of the element created by the merger are initialized when the element is deposited on the blackboard. For example, the strength of a new edge is set to the weighted average of the strengths of the two old edges; likewise, the grey-value of a new face is set to the weighted average of the grey-values of the two old faces. After the new element is created by the merger, any references to both of the old elements are replaced by a reference to the new element. Finally, if the two old elements were always referenced as a pair, a flag is set in the original objects indicating that each should be ignored in further processing; this flag is used because the newly created element subsumes the elements from which it was created.

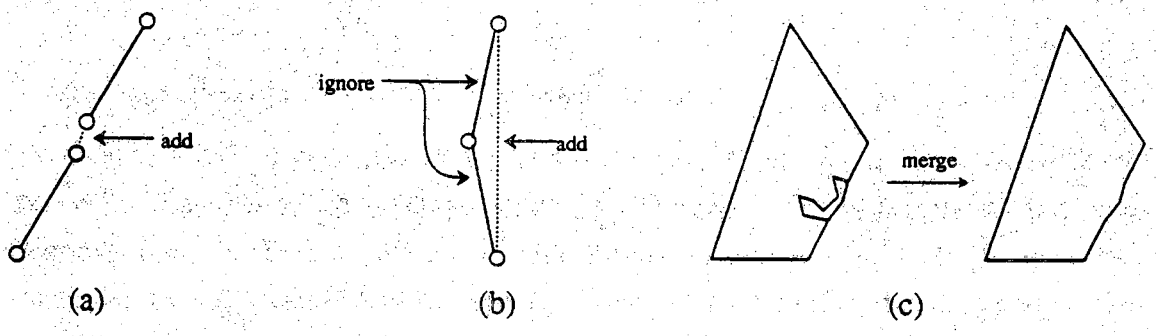


Figure 7.3 This figure shows the actions performed by the merger KS. Panel (a) shows how two close, collinear edges can be joined together. Panel (b) demonstrates how two collinear edges can be merged into a single edge. Finally, panel (c) shows how two adjacent face-elements with the same label can be merged if they are grouped together.

7.2.4. Operation of the Splitter Knowledge Source

The splitter KS also tries to correct the grouping of incorrectly grouped elements. However, it performs the opposite action of the merger KS; its task is to split data-elements into smaller elements if it is believed that they were incorrectly grouped. For example, it is also common for an initial grouping formed by the grouper KS to be contaminated by competing children; it is the splitter's task to correct these groupings. For example, when grouping edges into a face, the grouper may include multiple renditions of the same edge in a group. If the gray level variations corresponding to a scene edge do not exhibit a monotonic variation in the direction perpendicular to the edge, the the edge may be detected as multiple parallel edges in close proximity to one another. To check for these competing elements, the splitter is triggered whenever the grouper creates a new element.

Edges 1 and 24 in Fig. 7.2 could be an example of competing edges. The splitter detects parallel edges by measuring the angle and the extent of the overlap between two grouped elements with the same label. The overlap is measured by projecting the shorter of the edges onto the longer one. When such competing parallel edges are found, multiple groupings are formed from an initial group by retaining only one competing parallel edge at a time.

In the above example, edge 24 will compete with edges 1 and 2 in F_3; the same will be the case with the edges 6 and 9 in F_4. So, the above initial groups lead to the following groups:

$$F_1 = \{E_{16}, E_{18}, E_{19}\}$$
$$F_2 = \{E_3, E_4, E_8, E_{10}, E_{11}\}$$
$$F'_3 = \{E_1, E_2, E_{14}, E_{15}, E_{20}, E_{21}, E_{23}\}$$
$$F''_3 = \{E_{14}, E_{15}, E_{20}, E_{21}, E_{23}, E_{24}\}$$
$$F'_4 = \{E_5, E_7, E_9, E_{12}\}$$
$$F''_4 = \{E_5, E_6, E_7, E_{12}\}$$

Note that the splitter KS and the merger KS do not delete elements that they believe to be incorrectly grouped; instead, they create new elements and set a flag in the old element indicating that the element is no longer in focus. The old elements are not destroyed because the KSs may need to check if a newly created element is identical to an older element that is no longer in focus. The new element is deleted immediately if it is determined to be a duplicate. The older elements also are allowed to remain on the blackboard because, at some later time in the processing, it may be decided that they were correct and should be used.

CHAPTER 8

BLACKBOARD IMPLEMENTATION IN OPS83

Philosophically, all blackboard (BB) systems are alike in that they contain three main components. First, they all contain a collection of knowledge sources (KSs) into which the domain knowledge is partitioned; that is, each KS is able to solve a small portion of the total task. Second, blackboard systems are so named because each contains a blackboard, a hierarchical global database containing the data for the specific problem on which work is being done. To keep the KSs independent, communication between them is allowed to take place only through the blackboard database. Finally, each of the systems contains a control mechanism, commonly called the scheduler, that can respond opportunistically to data residing on the blackboard in order to optimize control flow.

Although all blackboard systems are conceptually similar, implementation details affect control strategies, KS granularity, etc. This chapter will address PSEIKI's implementation in OPS83 and the effects of the rule-based programming language on design decisions. The chapter will show the working memory data structures used for representing the data-elements on the BB and the knowledge source activation records (KSARs). Next, the current implementation of the scheduler and the monitor will be described. Finally, KS implementation will be described; the operation of the grouper KS will be described in detail and the operation of the labeler KS, the splitter KS and the merger KS also will be discussed.

8.1. Introduction to OPS83

OPS83 is the latest member of the OPS family of production system languages[†]. OPS83 contains many of the architectural features of its ancestors. As with the earlier OPS languages, OPS83 contain a set of unordered conditional statements called *production rules*. Each of these rules has two parts. The rule's antecedent is expressed as a collection of condition elements (CEs) on the rule's Left Hand Side (LHS); the LHS is used to match data elements stored in a global knowledge base called the *working memory*. The Right Hand Side (RHS) of a rule contains the consequent; the actions contained in the RHS are executed when the rule is fired. Although OPS83 is recognizably related to its ancestors; there are a number of significant differences between it and the earlier OPS languages.

OPS83 allows the programmer to encode knowledge using the procedural paradigm of traditional languages, such as Pascal and FORTRAN. Both procedures and functions can be called from the RHS of rules; they can be written either in the native OPS83 dialect or in foreign languages such as C, Pascal or FORTRAN. Furthermore, OPS83 allows the use of the common control structures, *if-then-else, while, for* and *return*, in the procedures, functions and RHS of rules. Although the OPS5 RHS actions *call* and *compute* allow the programmer to exploit some of the procedural and functional aspects of the underlying LISP environment, they are very limited compared to the integrated facilities provided in OPS83. In addition to their use in the RHS of rules, functions can also be used in the LHS of rules to match working memory elements (WMEs). These functions must be free of side effects; thus, they must not reference global variables (variables defined outside any rule, procedure or function), perform any input/output, modify the working memory or call another function with side effects.

OPS83 was developed to allow the development of fast, compact production system programs. To achieve this goal, the source code for OPS83 programs is compiled directly into machine language. Previous OPS languages were implemented in an interpreted LISP environment and the overhead associated with LISP limited their execution speed. One obvious consequence of compiling programs is that they cannot be changed at run time; thus; new rules cannot be defined using the *build* RHS action of the previous

[†] If not already familiar with the OPS family of languages, particularly OPS5, the reader is referred to [BroFar85] for a nice exposition on production system architectures. Version 2.2 of OPS83 has been used to program the current version of PSEIKI.

OPS languages. Although this fundamental limitation may interfere with the porting of some OPS5 machine-learning programs to OPS83; it has not had an effect on PSEIKI's development. Another consequence of the stand-alone nature of OPS83 programs is the absence of many of the programming environment utilities that OPS5 inherited from LISP. For example, there is no integral top-level shell in OPS83; however, a number of routines are supplied with the system that permit the programmer to tailor the environment to the application. PSEIKI can be run in an interactive mode with many of the abilities provided to OPS5 by the LISP environment. For example, WMEs can be added, modified, deleted and examined; the conflict set and a production's matches can be examined, and rules can be fired and traced. However, there are a number of OPS5 top-level commands that cannot be emulated in OPS83. For example, the *p*, *excise* and *pm* commands to build, delete and examine rules cannot be emulated. It also is impossible to implement the *back* command to undo the effects of rule firings.

Type checking in OPS83 is strongly enforced. Whereas fields of OPS5 WMEs were allowed to assume either symbolic or numeric values, the types of OPS83 variables and WME fields must be set at compile time. Integers, real numbers, logicals, characters and symbols are OPS83's atomic types; compound data types consist of arrays and structures. OPS5 WMEs are limited to having a single vector element; OPS83 WMEs can contain any number of structure or array fields. Furthermore, OPS83 contains no facilities for dynamic allocation of data structures; thus, the only way to store arbitrary length data structures is to store their components in WMEs (for example, a length N linked-list could be stored in N WMEs).

There are no conflict resolution strategies built into OPS83, whereas the LEX and MEA strategies were integral to OPS5. However, a number of routines are supplied to allow the user to program a custom strategy for each application. Unfortunately, it is not possible to implement the full OPS5 MEA conflict resolution strategy using the supplied routines. A sample conflict resolution routine is supplied with the OPS83 system, this sample resolution procedure was modified slightly for use in PSEIKI. The final conflict resolution strategy used in PSEIKI is shown below.

1) Refraction is used to eliminate all rule instantiations that have already fired.

2) The remaining instantiations are ranked based on the recency of the WME matching their first condition element. Only the instantiations containing the highest ranked (most recent) WME are allowed to remain in the conflict set.

3) The instantiations remaining after step 2 are then ranked based on the number of condition elements in the corresponding rule. Only the instantiations containing the largest number of condition elements are allowed to remain in the conflict set.

4) If multiple instantiations remain in the conflict set, then the instantiations are ranked based on a rating variable that can be associated with each production. Only the instantiations with the largest valued ranking variables are allowed to remain in the conflict set. The default value of this ranking variable is 0.0.

5) From set of instantiations remaining after step 4, the instantiation that was added to the conflict set most recently is chosen.

This procedure of ranking instantiations differs form the OPS5 MEA strategy because the specificity of the rule is ranked only on the number of condition elements present, not the total number of relational tests.

A number of guidelines for maximizing the efficiency of OPS programs were presented in [BroFar85]. In addition to these general guidelines, a number of programming strategies were used to make PSEIKI as efficient as possible. An obvious strategy for increasing the efficiency of a production system is to change the working memory as little as possible; whenever an WME is added, modified or deleted from the working memory, its change to the working memory must be propagated through the Rete network. PSEIKI's KSs keep from modifying the working memory excessively by storing their intermediate results in global variables. For example, the labeler KS stores an element's FOD and its initial belief function in global variables until they have been determined completely. Thus, only the global variables need to be updated as new elements are added to the FOD or as probability masses are determined. The labeler modifies the blackboard element being labeled only after the complete FOD and belief function have been determined. Likewise, the grouper KS stores the set of elements meeting the grouping constraint in a global list. The grouper builds a new element on the BB only after it has determined all of its children; storing the intermediate list of children in a global list prevents the grouper from repeatedly modifying an element as each new element is added to the list of its children. In most cases, storing intermediate results in global variables allows the KS to modify the elements in working memory only once per invocation. Because the WMEs are not modified until the end of KS processing, this strategy has the added benefit of allowing refraction to be used extensively to guide the flow of control inside the KS.

Other strategies have also been used to make PSEIKI as computationally efficient as possible. For example, because functions used in the LHS of rules are called by the Rete matcher whenever the working memory changes, they must be as efficient as possible. Sets of integers (such as the ID numbers of elements on the BB) in PSEIKI are implemented as sorted lists; thus, the test for inclusion in the set is performed as a binary search. Although the efficiency of the search could be increased by implementing the sets using key transformation (hashing) techniques, the scattering of the numbers through the data structure prevents the set from being enumerated easily. It also is helpful to screen possible LHS matches by placing relational tests before the function call in a condition element. For example, when matching the children of an element on the LHS of a rule, it is advantageous to test whether the elements are on the correct level of the BB before testing whether their ID numbers are in the parent's list of children.

8.2. OPS83 Data Structures Used By PSEIKI

PSEIKI uses the working memory of OPS83 for the BB data structure; each WME corresponding to the BB data structure describes a data-element at some level of the BB. In addition to being a host for the BB data structure, the working memory also stores the KSARs. Each KSAR holds the identity of the data-element that meets the triggering conditions of a KS, the relevant KS, and other pertinent information such as the cycle during which the KSAR was created. This information indicates to the KS the object on which work should be performed and aids the scheduler in choosing a KSAR to activate.

8.2.1. Working Memory Elements for Representing Data

A single WME class is used to store all non-vertex data elements on both the data panel and the model panel. That is, the same WME class is used for edges, faces, objects and scenes. Storing all BB elements in the same data structure allows generic functions to be applied to elements from all of the data levels. Fig. 8.1 shows the definition of the WME class for representing data.

Most of the WME fields are self-explanatory. The element's *id* number is a unique identifier used to keep track of individual data-elements; elements on the BB are always referenced via their ID numbers. The *panel* and *level* fields specify the element's location on the BB. The *panel* field specifies if the element is on the *model* panel or the

```
type Data=element (
    id:         integer;    -- unique ID number
    panel:      symbol;     -- type of panel (two_d or model)
    level:      integer;    -- Level in the panel
    source:     symbol;     -- source of data (grouper, etc.)
    seed:       integer;    -- seed element of a group
    children:   list;       -- children of element
    madeof:     array(2: integer);
    focus:      integer;    -- 1 if in focus set

    -- physical attributes of the element
    value:      integer;    -- edge strength, etc.
    size:       integer;    -- # of pixels in edge, etc.
    centroid:   vector;     -- centroid of the element's convex hull
    near:       vector;     -- extent of the convex hull
    far:        vector;

    -- label information
    frame:      list;       -- frame of discernment
    bpa:        bpas;       -- basic prob assignment
    label_status: integer-- status of element's label
                            -- this should equal
                            -- @.status[position(@.label, @.frame)]
    provided:   list;       -- elements used to update this element
    status:     list;       -- updating evidence focused on this
                            -- label already has been accumulated
    updating:   bfod;       -- updating bpa
    label:      integer;    -- label of element
    belief:     real;       -- belief in label
);
```

Figure 8.1 The Data WME class stores non-vertex BB elements.

two_d data panel. The *level* field specifies the level on which the element resides. Each level is assigned a unique integer (higher levels have larger numbers); thus, it is possible to test if an element is on or above a specific level of the BB by testing if the element's level field is larger than a specific constant. The *children* field is used to store the list of ID numbers of the element's children. The *source* field is used to specify how the element was created. If the element was deposited on the BB by the preprocessor, then its source value will be set to *original*; however, if the element was created by a KS, then the KS's name will be stored in this field. If the grouper KS was used to create the element, then the ID number of the child element that was used as the element's seed is stored in the *seed* field. The *made_of* field also is used to specify information about the element's genesis. If the element was created by the merger KS, then the ID numbers of the two elements that were merged to create the new element are stored in this field. If the element was created by the splitter KS, then the ID number of the element that was split to create the new element will be stored in this field.

The *focus* field has two functions. If the element is on the data panel, then this field is used as a flag indicating whether the element is in focus; a zero value indicates that the element is no longer in focus and should not be used in further processing. If the element is on the model panel, then this field is used to specify the desired number of instantiations for the model element. For example, if a model element's focus field is set to three, then the scheduler will try to fire the grouper KS and labeler KS to create three data elements with this element as their model.

The next few fields specify the physical attributes of the element. The *value* field is a generic attribute in which a level specific value is stored. For example, it is used to specify the strength of an edge or the average gray level of a face. The *size* parameter also is generic; this parameter is used to specify the length, area or volume of an element if it is an edge, face, or object, respectively. The *centroid* field is used to specify the centroid of the element's convex hull if the element is a face or object. The *near* and *far* parameters are used to specify the two diagonal vertices defining the extent of the element.

The remaining fields shown in Fig. 8.1 hold the belief function and related information for data elements. The *frame* attribute holds the set of all the ID numbers of the model elements in the data element's frame of discernment; the FOD is represented as an ordered list of integers. The *bpa* attribute holds the element's basic probability assignment. The BPA is represented as a set of N simple evidence functions (SEFs); each SEF

is represented as two real numbers -- the probability masses focused on the singleton proposition and its compliment. The probability mass focused on the FOD can be computed from these two values. The SEFs are ordered such that the i^{th} SEF corresponds to the i^{th} member of the element's FOD. The next field, *label_status*, is used to store the status of the element's label. Upon initialization of a label, this field is set to one. If the element's label had been updated, then the field is set to two. Finally, if the element's descendent's updating belief has been propagated up the hierarchy into the element's belief function, then the field is set to three. The next two attributes are used to guarantee that no element is used more than once to provide updating evidence for another. The *provided* field stores the ID numbers of the elements that have been used to provide evidence to update this element's belief function. The *status* list stores the *label_status* for each member of the element's FOD. This field is used to prevent the labeler KS from generating updating evidence for an element's label more than once. This field is needed because an element may have its label changed from its initial value to another value and back again. If the label_status information for each member of an element's FOD was not stored, then it would be possible to provide updating evidence for the element's initial label when it was first assigned and later when it was reassigned. The *updating* attribute is used to store the updating SEF. Finally, the element's *label* and *belief* information are stored in the next two attributes.

Another data structure is used to store information about vertices. This structure is shown in Fig. 8.2. As with the other data elements, the vertex's unique identifier is stored in its *id* field and the panel on which it resides is stored in its *panel* field.

```
type Vertex=element (
    id:         integer;    -- unique ID number
    panel:      symbol;     -- type of panel (two_d, model)

    rowcol:     ivector;    -- image coordinate of vertex
    coord:      vector;     -- world coordinate of vertex
);
```

Figure 8.2 The Vertex WME class stores vertex-level BB elements.

The next two parameters specify the vertex's location. The *rowcol* attribute indicates a vertex's coordinate on the image plane if it was obtained from 2D data. Likewise, the *coord* attribute specifies the vertex's location after it has been back-projected into 3D world coordinate frame.

8.2.2. The WME Class for Representing KSARs

Fig. 8.3 shows the WME class definition for representing a KSAR. The *id* field is used to keep track of the KSARs while the *status* field stores the state of a KSAR. The *KS* and *action* fields of the KSAR specify what action is to be performed on its focal-element. The *object* field is used to specify the ID number of the KSAR's focal element; the *level* and *panel* fields specify the location of the focal element on the BB. The *using* field specifies the secondary focal element. For example, when the merger KS is to merge two elements, the ID number of the second element is stored in this field. PSEIKI's scheduler uses the *priority* field when ranking KSARs for firing; only the KSARs for the splitter KS and merger KS have non-zero priority. The *trigger_cycle*, the *trigger_KSAR* and the *active_cycle* fields are used as a log of the BB activities;

```
type KSAR=element (
    id:             integer;    -- KSAR ID #
    status:         symbol;     -- KSAR status

    KS:             symbol;     -- Knowledge source being triggered
    action:         symbol;     -- action KS is to perform

    object:         integer;    -- Object being focused on
    level:          integer;    -- Level being focused on
    panel:          symbol;     -- Panel being focused on
    using:          integer;    -- Secondary object being focused on

    priority:       real;       -- KSAR priority.
    trigger_cycle: integer;     -- cycle KSAR was formed
    active_cycle:  integer;     -- cycle during which KSAR was active
);
```

Figure 8.3 The KSAR WME class stores Knowledge Source Activation Records.

they are used to record the BB cycle during which a KSAR was created, the KSAR which was active when the this KSAR was created and the BB cycle on which this KSAR was run, respectively. This information has proven useful for debugging the BB.

The KSAR originally is created with its status marked as *pending*. This means that the KS has been triggered but has not yet been run. When the scheduler decides to fire on a KSAR, it marks the KSAR's status to *active*. At this point, the KS's precondition and poisoning productions are allowed to fire; it is their job to mark the KSAR's status to *running* if the preconditions are met or to *poisoned* if they aren't. If the KSAR is determined to be poisoned, the KS's body productions are not allowed to fire and control is passed back to the scheduler. If the status has been set to running, the KS's body productions are allowed to fire. After the KS has accomplished its goal, it marks the KSAR's status field to *finished* and returns control to the scheduler.

8.2.3. Other WME Classes

A variety of other WME classes have been defined for use in PSEIKI. The *Panel* WME class is used to store information about the model and data panels of the BB. The fields of this class contain the type of data stored on the panel, the name of the file containing the panel's input data and the number of elements contained in the file. The WME Panel class definition is shown in Fig. 8.4.

```
type Panel=element (
    type:       symbol;     -- type of data (model, two_d)
    file_name: symbol;      -- name of file containing initial data
    elements:  integer[5];-- # of initial elements on a level
);
```

Figure 8.4 This is the WME class definition for a panel of the blackboard.

The *Level* WME class (Fig. 8.5) holds constants for the BB level specified by its *level* field. The next three fields are used by the labeler KS during the initialization, updating and propagation phases, respectively. The *evidence_scale* field specifies the factor, $SF_{initial}$, used to scale the amount of initial evidence for an element's label. The *update_scale* field specifies the factor, SF_{update}, used to scale the amount of updating evidence for an element's label. The *update_kids* field specifies the factor, $SF_{propagate}$, used to scale the amount of evidence provided by an element's descendents for its label. If the size of an element is less than the value in the *small* field, then it is considered to be too small to provide strong evidence about its siblings' labels and the updating evidence based on its label is reduced. Finally, the *group_thresh* field stores the threshold value used to determine if two elements should be grouped into an element on the specified level.

```
type Level=element (
    level:              integer;    -- level number
    evidence_scale:     real;       -- scale all initial evidence
                                    -- generated by the level's
                                    -- metric by this amount
    update_scale:       real;       -- scale all updating evidence
                                    -- generated by the level's
                                    -- metric by this amount
    kids_scale:         real;       -- scale all evidence generated
                                    -- by an element's descendent's
                                    -- by this amount
    small:              integer;    -- scale evidence if an element
                                    -- is smaller than this value
    group_thresh:       real;       -- element's won't be grouped if
                                    -- the grouping metric produces
                                    -- a value below this threshold
);
```

Figure 8.5 The Level WME class hold level specific constants.

Convex hull information for model elements is stored in the *Hull* WME class (see Fig. 8.6). The polygon defining the model element's convex hull is stored as a list of vertices following the hull in a counterclockwise direction. This WME class also stores the area of the hull and the diagonal vertices defining the hull's extent. The convex hulls for data elements are not stored in the working memory because the labeler only needs an element's convex hull when it is initializing the element's label. Thus, space can be saved, without any extra computational cost, by generating this hull information when it is needed.

```
type Hull=element (
    id:         integer;    -- unique id number
    hull:       polygon;    -- convex hull of the element
    area:       real;       -- area of the convex hull;
    near:       vector;     -- extent of the convex hull
    far:        vector;
);
```

Figure 8.6 The Hull WME class holds a model element's convex hull.

The homogeneous transformation matrices used by the labeler KS to update the belief in the labels of data elements are stored in the *Model_xfrm* WME class (see Fig. 8.7). These matrices specify the rigid motion transformations that make model elements compatible (see section 6.3). The *from* and *to* fields specify the ID numbers of the element providing the updating evidence and the element whose label is being updated, respectively. The transformation matrices are stored in the *xfrms* field. Two transformations are needed on the edge level of the BB (one collapses the edges, the other unfolds them -- see section 6.3.2); only one is needed on higher levels.

Two other WME classes are used in PSEIKI, both WME classes enable the use of well known production system techniques. The *Context* class is used to determine the flow of control using the OPS83 MEA-like conflict resolution strategy. WMEs of this class are used to enable rules pertaining to the current goal, function or subroutine by

```
type Model_xfrm=element (
    from:  integer;    -- ID number of model element 1
    to:    integer;    -- ID number of model element 2
    xfrms: transforms;-- homogeneous transformation matrices
);
```

Figure 8.7 The Model_xfrm WME class stores the homogeneous transformation matrices need by the labeler KS to update the belief in the labels of data-elements.

matching the first condition element of the appropriate rules. The *Constant* WME class is used to hold numeric, symbolic or logical constants. This class is needed because the LHS of OPS83 rules are not allowed to reference global variables; thus, constant information needed in the LHS of rules must be stored in WMEs.

8.3. Scheduler and Monitor Implementation

8.3.1. Scheduler Implementation

PSEIKI's scheduler consists of a set of metarules that run by default; that is, the scheduler runs automatically when no KS is active. Initially, when data is deposited on the BB, the scheduler is invoked to get the entire process started. As mentioned in chapter 7, PSEIKI's scheduling strategy is broken into phases: the *initialization* phase, the *updating* phase and the *propagation* phase.

The rules shown in Figs. 8.8 and 8.9 are used by the scheduler to initialize the labels of data elements during the initialization phase of BB processing. The rule shown in Fig. 8.8 is used to create the sub-goals to find the appropriate number of instantiations for each child of the model element. Presumably, the model element's children's instantiations then could be grouped into a data element and become an instantiation for the model element. This rule works as follows: The first two CEs are used to match the model element for the current goal. The third CE checks to see if there is a KSAR to label an element on the same level as the current goal element; the rule will not fire if

```
--
-- RULE: schedule_init_children
--   IF: We are looking for an instance of a model element
--     :     (a data element that can assume a model element's
--     :      ID number as its label)
--     : But we cannot find a data element on the correct level.
-- THEN: Create contexts to find instances of each of the model
--     :      element's children
--
rule schedule_init_children {
    &contxt (Context current=sched_init_model_instance);
    &model  (Data id=&contxt.object);
            (KSAR KS=label; action=initialize; level=&model.level);
-->
    local &i: integer;

    for &i = (2 to &model.children[1])
        make (Context current=sched_init_model_instance;
                      object=&model.children[&i];
                      level=&model.level-1);
};
```

Figure 8.8 This is the rule that chains down the model hierarchy creating context WMEs holding goals to find the descendents of a model element.

there is such an element. If there is data element on the current level, then the labeler KS should be fired to label it and the rule shown in Fig. 8.8 need not fire. When the rule fires, the RHS creates a context element (sub-goal) to find the instantiations for each child of the model element.

When a sub-goal is created to find an instantiation for a model element on a level that contains data elements, the rule shown in Fig. 8.9 becomes enabled and fires the labeler KS to initialize the labels of the elements on this level. This rule fires once for every data element on that level of the BB. The first two CEs of this rule are used to match a labeler KSAR on the correct level. The last two CEs are used to prevent the rule from firing if a priority KSAR is pending (the scheduling algorithm used to fire priority KSARs will be described later). This rule's only action is to fire the labeler KS on the element specified by the matched KSAR. Note that only the highest level elements deposited onto the data panel are labeled at this time (e.g. faces for a region-based preprocessor); the labels for elements on the lower levels than this are not initialized until the updating phase of BB processing. For example, the belief functions of edges are not

```
--
-- RULE: schedule_init_labels
--   IF: We are looking for an instance of a model element
--     : And there are KSARs available on this level.
--     : And no priority KSAR is pending.
-- THEN: Fire one of the appropriate KSARs.
-- NOTE: This rule will fire for each valid labeler KSAR until
--     :     all valid KSARs have been fired.
--
rule schedule_init_labels {
    &contxt (Context current=sched_init_model_instance);
    &ksar   (KSAR level=&contxt.level;
                  KS=label; action=initialize; status=pending);
    &thresh (Constant name=interrupt_threshold);
    -       (KSAR priority > &thresh.real_value; status=pending);
-->
    modify &ksar   (status=active; active_cycle=&current_cycle);
};
```

Figure 8.9 This rule is used to schedule the labeler KS to initialize the labels of data elements.

initialized until the updating phase if a region-based preprocessor was used to generate the input vision data.

After all of the labels for these elements have been assigned, the grouper is scheduled to group them into elements on higher levels of the BB. Another rule much like the one shown in Fig. 8.9 is used to determine that the grouper KS should be fired, other rules fire to determine the element that will be used as the group's seed-element. These rules rank the grouper KSARs based on belief in the element's label. Fig 8.10 shows one of the rules used to rank the grouper KSARs. This rule finds candidate elements that may be used as the seed element of the group; each candidate has one of the desired model element's children as its label (i.e. they are instantiations of the model's children). It finds one of these candidate elements for each of the children of the model element being formed. The first two CEs guarantee that a candidate element with a particular label is found. The third CE matches the data element that will become the candidate seed element. The fourth CE guarantees that the candidate has not been used as the seed element for another group; in effect, this prevents an element from being the seed element for more than one group. Finally, the last CE matches the grouper KSAR with

```
--
--    RULE: find_group_candidate
-- ACTION: Choose as seed candidates the data elements with
--       :   highest belief that correspond with each of
--       :   the model element's children.
--      IF: We are looking for grouper KSAR seed candidates by
--       :   looking at a model element's kids
--    THEN: Select the data element corresponding to the kid that
--       :   has the highest belief and mark the corresponding
--       :   KSAR as a candidate
--
rule find_group_candidate {
    &contxt (Context current=sched_find_candidate);
    &m_kid  (Data id=&contxt.using);
    &el     (Data label=&m_kid.id);
    -       (Data label=&contxt.object; seed=&el.id);
    &ksar   (KSAR object=&el.id; KS=group; action=initialize;
                  status=pending);
    [&el.belief];
-->
    modify &ksar   (status=candidate);
    remove &contxt;
};
```

Figure 8.10 This is one of the rules used to rank grouper KSARs.

the designated seed-element. The structure with the square brackets on the next line is the production's ranking variable; as mentioned in section 8.1, OPS83 uses this value to rank instantiations in the conflict set. Everything else being equal, OPS83 will fire the instantiation with the greatest value for the expression in the brackets. Thus, the construct will force the rule to fire on the data element with the largest belief. When this rule fires, it flags the KSAR as a candidate and deletes the context so that the rule will not fire again. Other rules are used to select the optimum candidate as the seed element and fire the grouper KS; most are very similar to the one just discussed and are not shown here for brevity's sake.

After the initialization phase of processing is complete, the scheduler starts the updating phase. The rule shown in Fig. 8.11 fires the labeler KS to update the belief in an element's children's labels. The first two CEs in the LHS of this rule are used to match the KSAR to be fired. The last two CEs of this rule are used to prevent the rule from firing if a priority KSAR is pending. The RHS of this rule changes the status of the

```
--
-- RULE: fire_on_update_element
--   IF: We are trying to update the belief in the labels of
--     :     an element's kids and there is a KSAR pending to do it.
--     : And no priority KSAR is pending.
-- THEN: Fire on the KSAR
--
rule fire_on_update_element {
    &contxt (Context current=sched_update_kids);
    &ksar   (KSAR object=&contxt.object;
                  KS=label; action=update_kids; status=pending);
    &thresh (Constant name=interrupt_threshold);
    -       (KSAR priority > &thresh.real_value; status=pending);
-->
    modify &ksar   (status=active; active_cycle=&current_cycle);
};
```

Figure 8.11 This rule is used to fire the labeler KS during the updating phase of BB processing.

labeler KSAR to active, causing the KS to fire.

The following rule (Fig. 8.12) will fire after the beliefs in an element's children's labels have been updated; this rule generates a context to force the scheduler to fire the labeler KS to update the belief function of each of the element's grandchildren. Thus, it forces the scheduler to chain down the data hierarchy firing the labeler KS to update the beliefs in the element's descendent's labels. The first two CEs of this rule prevent the rule from firing until the labeler is fired to update the belief in the element's children's labels. The third CE matches the parent element. In the RHS, a context is generated to fire the labeler KS to update the belief in the labels of each of the parent element's children. Finally, the context is removed to prevent the rule from firing more than once.

Because the monitor creates propagation phase labeler KSARs only for scene level elements, the scheduler picks propagation KSARs at random until there are none pending. The rule shown in Fig. 8.13 is used to fire the labeler KS with a propagation KSAR. The first two CEs of this rule match the pending propagation KSAR. The last two CEs are used to prevent the rule from firing if a priority KSAR is pending. When the rule fires, it marks the KSAR's status to active, firing the KS.

```
--
-- RULE: schedule_update_children
--   IF: We just updated the labels of an element's kids
-- THEN: Make contexts to update the kids of each of them
--     :     (i.e. the element's grandkids).
-- NOTE: This rule fires AFTER "fire_on_update_element"
--
rule schedule_update_children {
    &contxt (Context current=sched_update_kids);
    &ksar   (KSAR object=&contxt.object; level>FACE;
                  KS=label; action=update_kids; status<>pending);
    &el     (Data id=&contxt.object);
-->
    local &i: integer;

    for &i = (2 to &el.children[1])
        make (Context current=sched_update_kids;
                                  object=&el.children[&i]);
    remove &contxt;
};
```

Figure 8.12 This rule makes the scheduler fire the labeler KS to update the labels of elements on the lower levels of the BB.

The KSARs for the splitter KS and the merger KS have a higher priority than those of the labeler KS and the grouper KS. If one or more of these priority KSARs are created during the course of BB processing, then the scheduler will rank them based on their priorities and fire the KSARs with the highest priority. The scheduler will return to the normal BB scheduling process after all of the priority KSARs have been fired. If two or more KSARs have the same maximum priority value, then one is selected at random. The scheduler will continue to fire these KSARs until there are no pending splitter KSARs or merger KSARs with a priority greater than a predefined threshold left in working memory. Note that the priority field is always zero for labeler KSARs and grouper KSARs because they are scheduled using the scheme described above. The rule shown in Fig. 8.14 selects the KSAR with highest priority and fires the appropriate KS.

```
--
-- RULE: schedule_propagate
--   IF: The current context indicates that we should pass
--     :     belief up the hierarchy for a particular element.
--     : And there is a pending KSAR to do that
--     : And no priority KSAR is pending.
-- THEN: Fire on the KSAR
--
rule schedule_propagate {
    &contxt(Context current=sched_incorp_element);
    &ksar   (KSAR object=&contxt.object;
                  KS=label; action=incorp_update; status=pending);
    &thresh(Constant name=interrupt_threshold);
    -       (KSAR priority > &thresh.real_value; status=pending);
-->
    modify &ksar   (status=active; active_cycle=&current_cycle);
};
```

Figure 8.13 This figure shows the rule used to fire the labeler KS to incorporate, into an element's belief function, the belief generated by checking the consistency of the element's descendent's labels.

```
--
-- RULE: schedule_fire_interrupt
--   IF: There is at least one pending KSAR with
--     :     non-zero priority
-- THEN: Fire on the pending KSAR with the highest priority
--
rule schedule_fire_interrupt {
    &ksar   (KSAR priority>0.0; KS<>label; status=pending);
    -       (Constant name=interrupt_threshold;
                      real_value > &ksar.priority);
    -       (KSAR priority > &ksar.priority; status=pending);
    -       (KSAR status=running);
    -       (KSAR status=active);
-->
    modify &ksar   (status=active; active_cycle=&current_cycle);
};
```

Figure 8.14 This rule is used to fire the splitter KS or merger KS.

8.3.2. Monitor Implementation

The BB monitor makes extensive use of OPS83 demons. A demon in OPS83 is a rule whose first CE is not a context or KSAR. Because of the OPS83 rule selection strategy, these rules take precedence over ordinary rules (e.g. rules inside of KSs or scheduler rules) and fire as soon as they become completely instantiated. Thus a demon in OPS83 can be thought to operate in the background outside of any context, KS or goal search.

Fig. 8.15 shows an example of a monitor rule for the grouper KS; the monitor rules for the other KSs are very similar. This rule fires when it detects a labeled data-element without any parents (an orphan element). The rule creates a KSAR directing the grouper KS to use the orphan as the seed element of a group. The first CE of this rule matches any data element with a label; this data-element is the *focus-element* of the rule. The second CE allows the rule to fire only if the preprocessor did not deposit any elements on a level higher than the orphan element.

```
--
-- RULE: group_trigger
--   IF: There is a labeled element that is being focused on but
--     : has not yet been placed in a group
--     : AND its level is greater than or equal to the highest
--     :     level that the preprocessor created.
--     : AND a KSAR saying that it should be grouped
--     :     has not yet been created
-- THEN: Create a KSAR that indicates that the element
--     :     should be grouped
--
rule group_trigger {
    &el (Data panel<>model; label<>0; focus<>0);
        (Constant name=highest_level; int_value <= &el.level);
    -   (KSAR KS=group; action=initialize; object=&el.id);
    -   (Data level=&el.level+1, in_list(&el.id, @.children));
-->
    make (KSAR KS=group; action=initialize;
               trigger_cycle=&current_cycle;
               id=&next_KSAR_id; status=pending;
               object=&el.id; panel=&el.panel; level=&el.level);
    &next_KSAR_id = &next_KSAR_id + 1;
};
```

Figure 8.15 This is a monitor demon that is used to create a KSAR for the grouper KS.

The third CE prevents the rule from firing if the grouper KS already has been triggered on this data-element. Finally, the last CE guarantees that the focus-element is an orphan. This CE uses the function *in_list()* to match any WME that has the first CE's ID number in its list of children.

The monitor rules for the merger KS are slightly different because of the two step triggering process described in chapter 7. The two rules that are used to trigger the merger KS to merge two faces are shown in Fig. 8.16. The first rule is used to partially trigger the merger for data elements on all levels of the hierarchy. It creates a merger KSAR with the status *partially_triggered* whenever it detects two elements with the same label (if there isn't already a KSAR to merge them). The second rule is used for only face-level elements. It checks to see if the two faces are adjacent and have non-zero belief in their labels. The fourth CE of this rule is used to prevent the rule from firing if an edge-based preprocessor was used to generate PSEIKI's input data. As mentioned in chapter 7, the merger is allowed to merge faces only if a region-based preprocessor was used.

8.4. Implementation of the KSs

Even though the various KSs perform very different tasks, many common subtasks are performed by all of them during KS operation. These subtasks start when the scheduler marks a KSAR's status to active. After a KS becomes active, the monitor's *poisoning* rules for that KS are allowed to fire; these rules are used to guarantee that the KS's preconditions have not become invalid since the KS was triggered. If a poisoning rule does fire, it sets the KSAR's status to poisoned and returns control to the scheduler. If none of the poisoning rules fire, a rule that marks the KSAR's status to *running* fires by default.

The flow of control becomes more KS specific after the KS starts running, but it still follows many of the same patterns. In most KSs, *driver rules* are the first few rules that fire at the start of KS processing. These rules don't contribute directly to the solution of the KS's task; instead, they initialize the elements that the KS needs to solve the task. These driver rules can generate contexts needed by the KS in its problem solving activity and build dummy BB data-elements that will be "fleshed out" during the course of the KS's processing. The driver rules also can initialize global variables used by KS. After the KS's driver rules are fired, the control flow becomes very KS specific.

```
--
-- RULE: merge_partial_trigger
--   IF: There are two elements that have the same label.
-- THEN: Create a merger KSAR with status "partially triggered"
--
rule merge_partial_trigger {
    &el1    (Data label<>0; panel<>model; focus<>0);
    &el2    (Data label=&el1.label; id<>&el1.id; focus<>0);
            (KSAR object=&el1.id; using=&el2.id; KS=merge);
-->
    make (KSAR KS=merge; action=merge_adjacent;
               trigger_cycle=&current_cycle;
               id=&next_KSAR_id; status=partially_triggered;
               object=&el1.id; panel=&el1.panel; level=&el1.level;
               using=&el2.id; priority=MERGE);
    &next_KSAR_id = &next_KSAR_id + 1;
};
```

```
--
-- RULE: face_merge_trigger
--   IF: There are two adjacent elements that have the same label.
--     : AND an edge-based preprocessor was not used.
-- THEN: Fire the merge KS to merge them
--
rule face_merge_trigger {
    &ksar   (KSAR KS=merge; status=partially_triggered; level=FACE)
    &face1  (Data id=&ksar.object; belief > 0.0);
    &face2  (Data id=&ksar.using;  belief > 0.0);
            (Constant name=highest_level; int_value>EDGE);
    &edge   (Data level=EDGE; in_list(@.id, &face1.children);
                              in_list(@.id, &face2.children));
-->
    modify &ksar   (status=pending);
};
```

Figure 8.16 These two rules are used to trigger the merger KS to merge two faces.

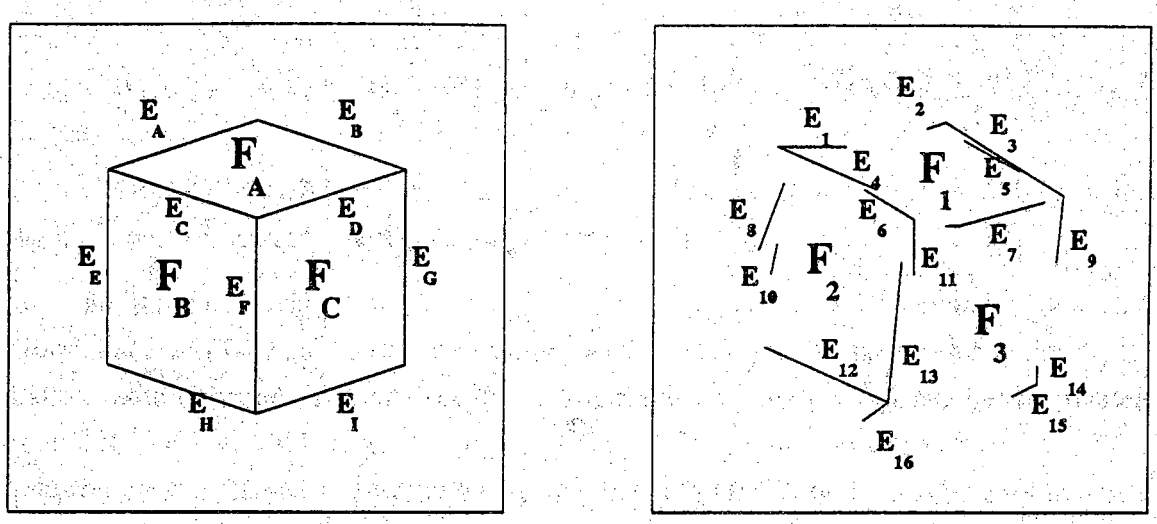


Figure 8.17 This figure shows an example of data on the BB and is used to explain KS operation.

8.4.1. Grouper KS Implementation In the next few sections, the control flow inside each KS will be demonstrated through the use of examples.

To illustrate the flow of control inside a KS, the grouper KS's formation of a face from edges will be examined. The example in Fig. 8.17 will be used to make the explanation more concrete. Assume for the example that the grouper KS has been activated with a KSAR focused on the element E_9 of Fig. 8.17. As previously described, the KS's poisoning rules are allowed to fire when it is first activated. Fig. 8.18 is an example of a rule that the monitor could use to poison a grouper KSAR. This rule is meant to poison a grouper KSAR if its seed element is already part of two or more groups[†]. This rule works in the following manner: The first CE matches the active grouper KSAR. The next two CEs try to find two faces that contain the edge. If these two CEs match two separate faces, then the rule fires and marks the KSAR as poisoned. If no poisoning rules fire, another rule fires by default and marks the KSAR's status as running. Thus if it is

[†] This does not imply that a data-element can participate only in a single group. An edge-element, for example, is allowed in two or more groups if it is on the common boundary between them. However, an edge-element can not serve as a seed if it already is part of two or more faces. Therefore, an edge-element that belongs to two or more groups can trigger the formation of only one of them; other edges would have to act as seeds for the other groups.

```
--
-- RULE: edge_group_poison
--   IF: The active KSAR focuses on an edge in more than one group
-- THEN: Poison the KSAR
--
rule edge_group_poison {
    &ksar   (KSAR KS=group; action=initialize; status=active);
    &parent (Data level=FACE; in_list(&ksar.object, @.children);
                  focus<>0);
            (Data level=FACE; in_list(&ksar.object, @.children);
                  focus<>0; id<>&parent.id);
-->
    modify &ksar   (status=poisoned);
};
```

Figure 8.18 This figure shows an example of a poisoning rule.

assumed that element E_9 has not been used as the seed for another group, the active KSAR's status is set to running.

The grouper KS uses a driver rule to initialize internal processing; this rule fires immediately after the KS starts running. The driver rule is used to initialize a global variable containing the new element's list of children; it also creates a context to continue grouping. Fig. 8.19 shows the driver rule for group initialization. The CEs in the LHS of this rule match the running KSAR, the seed element, and the proposed label element for the element being created, respectively. This rule will fire only once during any KS activation because none of these elements will be modified during KS processing, When the rule fires, a global variable containing the list of children is initialized with the seed element's ID number. Later in KS processing, this list will be copied into the new element's list of children. The rule then creates a context indicating that other elements should be grouped with the seed element.

In the example, this driver rule would fire because edge E_9 is an orphan. When the rule fires, the global list of children, *&kids*, is initialized to contain the seed element's ID number. After the driver rule initializes the child list, the remaining KS body rules can fire. Only one KS body rule needs to fire to include other edge elements in the group. This rule (shown in Fig. 8.20) fires once for every edge that can be grouped into the face.

```
--
-- RULE: group_driver -- start up grouping process by initializing
--     :              the global list of children
--     :           -- also make a context to continue
--     :              the grouping process
--   IF: There is a running KSAR that says to initialize a group
-- THEN: Set the global list of children to the seed element.
--     : And make a context to continue grouping.
--
rule group_driver {
    &ksar   (KSAR KS=group; action=initialize; status=running);
    &el     (Data id=&ksar.object);
    &model  (Data in_list(&el.label, @.children));
-->
    call clear_list(&kids);
    call add_element(&ksar.object, &kids);
    make (Context current=continue_grouping;
                  object=&el.id; using=&model.id);
};
```

Figure 8.19 This driver rule fires at the start of grouper KS processing.

The first three CEs of the rule in Fig. 8.20 find the active context, the desired label of the new face and the seed element, respectively. The fourth CE finds a candidate edge to group into the face. This CE allows only edges with correct labels to be grouped into the parent; it does this by checking to see if the candidate edge's label element is a child of the new face's desired label element. The rest of the CEs merely obtain data needed in the right hand side (RHS) of the rule. Four of them are used to match the vertices defining the endpoints of the seed edge and the candidate edge. The ninth CE matches a WME holding a homogeneous transformation matrix. The transformation matrix is defined to transform the seed edge's label-element so that it is compatible with the candidate's label-element. The final two CEs match constants used in RHS processing.

When the rule fires, the compatibility between the candidate and a transformed version of the seed element is computed as described in chapter 5. If this value is greater than a threshold, then the *add_element()* function is used to add the candidate's ID number to the parent's list of children. Notice that this rule changes nothing in the working memory and that refraction prevents the rule from firing again with the same instantiation.

```
--
-- RULE: group_into_face -- group edge-elements into a face
--   IF: We are grouping edges into a face and there is a
--     :   compatible edge that is not yet in the face
-- NOTE: An edge is compatible if its label element is a
--     :   sibling of the seed element's label element
-- THEN: See if the xformed version of the edge is collinear
--     :   with the focus element.  If so, put it into the group
--
rule group_into_face {
    &contxt (Context current=continue_grouping);
    &model  (Data id=&contxt.using; level=FACE);
    &edge1  (Data id=&contxt.object);
    &edge2  (Data panel<>model; level=EDGE; id<>&edge1.id;
                  focus<>0; in_list(@.label, &model.children));

    -- get parameters needed in rhs computations
    &s1     (Vertex id=&edge1.children[2]);
    &e1     (Vertex id=&edge1.children[3]);
    &s2     (Vertex id=&edge2.children[2]);
    &e2     (Vertex id=&edge2.children[3]);

    &xfrm   (Model_xfrm from=&edge2.label; to=&edge1.label);
    &slop   (Constant name=max_dist);
    &level  (Level level=EDGE);
-->
    local &compat, &incompat:  real;

    call edge_compatibility(&s1.coord, &e1.coord, &edge1.label,
                            &s2.coord, &e2.coord, &edge2.label,
                            &xfrm.xfrms, &slop.real_value,
                            &compat, &incompat);

    if (&compat > &level.group_thresh)
        call add_element(&edge2.id, &kids);
};
```

Figure 8.20 This rule is used to group edges into faces.

In the example, any edge that has one of the labels E_D, E_F, E_G or E_I is a candidate to be grouped with edge E_9 into face F_3. Edges E_7, E_{11}, E_{13}, E_{14} and E_{15} meet this criterion. Thus any of these edges that was compatible with the transformed version of the focus-element would be grouped into the parent. If all but E_{11} were compatible with the transformed E_9 then the children of F_3 would be edges E_7, E_9, E_{13}, E_{14} and E_{15}.

The rule shown in Fig. 8.21 is used to form the new element at the end of KS processing. This rule is enabled by the active grouper KSAR WME, but its LHS is very general so it doesn't fire if any other KS rules are enabled. In the example, the rule shown in Fig. 8.21 would create a new element, say face F_3, and deposit it on the BB. The new face's list of children would be copied from the global list of children initialized by the previous two rules. After the element was created, a context would be generated to check whether it was a duplicate of another face. Two elements are said to be duplicates if their lists of children are identical. Finally, the grouper would explicitly trigger the labeler to give the new element its initial label.

8.4.2. Labeler KS Implementation

In this section, some of the rules used by the labeler KS will be highlighted to demonstrate how it accomplishes its tasks. Rules to initialize an edge element's FOD and its belief function will be shown. The rule used to generate updating belief in one edge's label based on another's label also will be shown. There are many other rules in the labeler KS (equivalent rules that operate on other levels of the BB, rules used to control the flow inside the KS, etc). However, it is believed that the rules shown here form a representative sample of those that the KS uses to perform its task.

The rule shown in Fig. 8.22 is used to add model elements to an orphan data element's FOD. This rule fires once for every model element on the same level of the BB as the data element. The first two CEs in this rule are used to match the element whose label is being initialized. The third CE is used to prevent this rule from firing if the element has a parent. The fourth CE matches the model element that may be added to the data element's FOD. The last CE is used to retrieve the maximum amount of misregistration specified by the user, D_{max}. When the rule fires, a function is called to check whether the element's expanded extent overlaps the model element's extent. If so, the model element's ID number is added to the element's FOD list. If the element is not an orphan, then another rule is fired; this rule copies the list of children from the

```
--
-- RULE: group_end -- finish up grouping by actually making
--     :              the new parent element
--   IF: We can't find any more elements to group
-- THEN: Create a new parent element using the global child list
--     :       as the new element's child list.
--     : ALSO create a context to see if the new element is a
--     :       duplicate of another element and make a KSAR to
--     :       initialize the new element's label.
--
rule group_end {
    &ksar  (KSAR KS=group; action=initialize; status=running);
    &el (Data id=&ksar.object);
-->
    &max_id = &max_id + 1;
    make (Data id=&max_id; source=grouper; seed=&ksar.object;
               panel=&el.panel; level=&el.level+1;
               focus=1; label_status=UNLABELED;
               call copy_list(&kids, @.children));
    make (Context current=check_for_duplicate; object=&max_id);
    make (KSAR KS=label; action=initialize;
               trigger_cycle=&current_cycle;
               id=&next_KSAR_id; status=pending;
               level=&el.level+1; object=&max_id);
    &next_KSAR_id = &next_KSAR_id + 1;
};
```

Figure 8.21 This figure shows the rule used by the grouper to form the new data element.

```
--
-- RULE: initialize_orphan_fod
--   IF: The element whose FOD is being initialized
--     :     doesn't have a parent
-- THEN: Set the FOD to all model elements whose expanded extent
--     : overlaps the element's extent
-- NOTE: Rule fires at least once for every model element
--     :     on the same level as the data element
--
rule initialize_orphan_fod {
    &contxt (Context current=initialize_fod);
    &el     (Data id=&contxt.object);
            (Data level=&el.level+1,
                  in_list(&contxt.object, @.children));
    &model  (Data panel=model; level=&el.level; focus<>0);
    &max    (Constant name=max_dist);
-->
    if (expanded_overlap(&near, &far,
                         &model.near, &model.far,
                         &max.real_value)) {
        call add_element(&model.id, &init_fod);
    };
};
```

Figure 8.22 This rule is used to initialize the FOD for an orphan element.

element's parent's label element into the element's FOD list.

After the element's FOD has been determined, a level-specific rule is fired to initialize its belief function. The rule show in Fig. 8.23 is used to initialize the belief function of edge elements. When this rule fires, the edge is translated using the predefined homogeneous matrix, *xfrm*. If the edge has a parent, then the transformation is defined to make the centroids of the edge's parent face and that face's model element coincident. If the edge is an orphan, then the transformation is an identity transformation. After the edge is transformed, the expected scene (in)compatibility metrics defined in chapter 6 are used to determine the degree to which the edge matches the model element. This rule is fired once for every member of an element's FOD.

The rule shown in Fig. 8.24 is used to update the belief in one edge's label based on the labels of its siblings. The first three CEs of this rule are used to reference the elements in question. The second CE is the edge whose label is being updated and the third CE is the edge on which the new evidence is based. The remaining CEs are used to

```
--
-- RULE: initialize_edge_SEF -- initialize the SEF for one of the
--     :           model edges of a data edge
--   IF: There is a context to initialize the SEF for a member
--     :    of a data edge's FOD.
-- THEN: Compute the ES_compat and ES_non_compat for the edge and
--     :    the model edge and set the masses to those values.
--
rule initialize_edge_SEF {
    &contxt (Context current=initialize_bpn);
    &el (Data id=&contxt.object; level=EDGE);
    &model (Data id=&contxt.using; focus<>0);

    &el_s  (Vertex id=&el.children[2]);
    &el_e  (Vertex id=&el.children[3]);
    &mod_s (Vertex id=&model.children[2]);
    &mod_e (Vertex id=&model.children[3]);
    &slop  (Constant name=max_dist);
-->
    local &position: integer;
    local &belief, &disbelief: real;
    local &collin, &noncollin, &in_range, &not_in_range: real;
    local &xfrm_s, &xfrm_e: vector;

    call apply_xfrm(&xfrm_s, &xfrm, &el_s.coord);
    call apply_xfrm(&xfrm_e, &xfrm, &el_e.coord);

    call in_range(&mod_s.coord, &mod_e.coord,
            &xfrm_s,      &xfrm_e, &slop.real_value,
            &in_range, &not_in_range);
    if (&in_range > 0.0) {
        call ES_collinearity(&mod_s.coord, &mod_e.coord,
                    &xfrm_s,  &xfrm_e,
                    &slop.real_value,
                    &collin, &noncollin);
        &belief    = &collin * &in_range;
        &disbelief = &noncollin * &not_in_range;
    } else {
        &belief = 0.0;
        &disbelief = 1.0;
    };

    &position = position(&model.id, &init_fod);
    &init_bpa[&position].positive = &belief;
    &init_bpa[&position].negative = &disbelief;
    remove &contxt;
};
```

Figure 8.23 This rule is used to initialize an element's belief function.

match WMEs needed in the RHS of the rule. In particular, the fourth CE matches the CE defining the transformation needed to make the first edge's label element collinear with the second edge's label element. This element will not be matched if its belief is too weak or if it already has been used to update the belief in the first edge's label. When the rule fires, the compatibility of the second edge with a transformed version of the first edge is measured. Then the (in)compatibility values are scaled based on the second edge's belief, the edge-level scale factor and the length of the second edge. The new evidence is the accumulated into the first edge's updating belief function. Finally, The second edge's ID number is added to the list of ID numbers of edges already used to update the belief in the first edge's label.

8.4.3. Splitter KS and the Merger KS Implementation

The flow of control inside the splitter KS will be explored by examining the rules used to split a face with competing edges into multiple faces with one competing edge apiece. The splitter KS uses a driver rule to initialize processing; this rule is used to generate a context that directs the KS to examine the focus element for competing edges. After the driver rule fires, a level-specific body rule that finds all the competing children of an element is allowed to fire; this rule fires at least once for every pair children that could possibly compete. For example, the splitter KS uses the rule shown in Fig. 8.25 to find competing edges that the grouper has included in a face. When it finds a pair of competing edges, it creates two new faces each with only one of the competing edges; it also resets the focus flag in the original face to prevent its use in further BB processing. Finally, it generates a context for each of the new edges to determine if they also contain competing edges. The rule works as follows: The first two CEs are used to match the newly created face element; they also keep the rule from firing more than once. The second two CEs are used to match two child-edges with identical labels (only edges with identical labels can compete). When the rule fires, the function *edge_overlap()* determines the overlap of the two edges using the technique described in chapter 7. If the overlap is found to be greater than a preset threshold, then the two edges are considered to be competing, and the face is split into two faces with one competing edge apiece. Finally the rule generates two contexts to check the new faces for competing edges.

The merger KS does not require any driver rules. When the KS is fired by the scheduler, a level-specific rule is fired to merge the KS's focus element with the

```
--
-- RULE: update_edge_certainty -- use the belief in one edge's
--     :                           label to update the belief in
--     :                           one of its siblings
--   IF: There is a context to use one edge to update
--     :     the belief in another
--     : AND we haven't already used this one to
--     :     update the other one
-- THEN: Do so using the collinearity metric and add this
--     :     edge's ID # to the provided list.
--
rule update_edge_certainty {
    &contxt (Context current=generate_update_certainty);
    &el1    (Data id=&contxt.object; level=EDGE);
    &el2    (Data id=&contxt.using; belief > MIN_BELIEF;
                  ~in_list(@.id, &el1.provided));
    -- get parameters needed in rhs computations
    &xfrm   (Model_xfrm from=&el1.label; to=&el2.label);
    &level  (Level level=&el1.level);
    &s1     (Vertex id=&el1.children[2]);
    &e1     (Vertex id=&el1.children[3]);
    &s2     (Vertex id=&el2.children[2]);
    &e2     (Vertex id=&el2.children[3]);
-->
    local &pos, &neg, &scale: real;
    local &new_evidence: bfod;
    local &s1_xfrm, &e1_xfrm: vector;

    call edge_compatibility(&s2.coord, &e2.coord, &el2.label,
                            &s1.coord, &e1.coord, &el1.label,
                            &xfrm.xfrms,
                 distance(&s2.coord, &e2.coord),
                            &pos, &neg);
    &scale = scale_certainty(&el2.size, &level.small);
    &new_evidence.positive = &pos * &el2.belief
                             * &level.update_scale * &scale;
    &new_evidence.negative = &neg * &el2.belief
                             * &level.update_scale * &scale;
    call update_bfod(&accum_bpa, &new_evidence);
    call add_element(&el2.id, &provided);
    remove &contxt;
};
```

Figure 8.24 This rule is used to initialize an edge's belief function.

```
--
-- RULE: face_split -- split a face if it contains (at least)
--     :                two competing edges.
--   IF: There are two edges with the same label in the face
--     :     that we are trying to split
-- THEN: If the overlap of the smaller edge with the larger
--     :     is greater than some threshold, then split the
--     :     face into two faces, each with one of the edges.
--
rule face_split {
    &contxt (Context current=check_competing);
    &face   (Data id=&contxt.object; level=FACE);
    &edge1  (Data level=EDGE; in_list(@.id, &face.children));
    &edge2  (Data level=EDGE; in_list(@.id, &face.children);
                 id<>&edge1.id; label=&edge1.label;
                 size<&edge1.size);

    &longs  (Vertex id=&edge1.children[2]);
    &longe  (Vertex id=&edge1.children[3]);
    &shorts (Vertex id=&edge2.children[2]);
    &shorte (Vertex id=&edge2.children[3]);
-->
    if (edge_overlap(&longs.coord,  &longe.coord,
                     &shorts.coord, &shorte.coord) > THRESH) {
        modify &face (focus = 0);

        -- create two new faces,
        -- each with one of the competing edges.
        &max_id = &max_id+1;
        make (Data call duplicate_Data(&face, @);
                   id = &max_id; madeof[1] = &face.id;
                   call delete_element(&edge1.id, @.children));
        make (Context current=split_element; object=&max_id);

        &max_id = &max_id+1;
        make (Data call duplicate_Data(&face, @);
                   id = &max_id; madeof[1] = &face.id;
                   call delete_element(&edge2.id, @.children));
        make (Context current=split_element; object=&max_id);
    };
};
```

Figure 8.25 This rule is used to split a face with competing edges into multiple faces with one of the competing edges apiece.

secondary focus element. For example, the rule shown in Fig. 8.26 is used to merge two faces. The LHS side of this rule is used to match the two focus elements and guarantee that the rule will fire only once. The RHS of the rule builds an element on the face level of the data panel into which the two focus elements are merged. The children of this new element is set to be the exclusive-or of the children lists of the two focus elements. Then, a context is generated to determine whether the new face is a duplicate of a face already present on the BB. If the data element is a duplicate, its focus flag is reset. Finally, a context is generated to determine whether either of the merged faces' focus flags need to be reset. If one of the merged faces always is referenced along with the other merged face, then the face's focus flag is reset. Therefore, if the merged faces are always referenced as a pair, both of their focus flags will be reset. The context also enables a rule that replaces references to the old faces with a reference to the new face. If an element references both of the merged faces, then the two references are replaced by a reference to the new face.

```
--
-- RULE: face_merge -- merge two faces
--   IF: There is a KSAR to merge two faces and an
--     :     edge-based preprocessor was not used.
-- THEN: merge the two faces into a new face and
--     :     see if it is a duplicate
--
rule face_merge {
    &ksar   (KSAR KS=merge; action=merge_adjacent;
                  status=running; level=FACE);
    &el1    (Data id=&ksar.object; focus<>0);
    &el2    (Data id=&ksar.using; focus<>0);
       (Constant name=highest_level; int_value>EDGE);
    -  (Data madeof[1]=&el1.id; madeof[2]=&el2.id);
-->
    &max_id = &max_id + 1;
    make (Data id=&max_id; source=merger; focus=1;
               panel=&el1.panel; level=&el1.level;
               size=&el1.size+&el2.size;
               value = weighted_average(&el1.value, &el1.size,
                                        &el2.value, &el2.size);
               label_status=UNLABELED;
               call near_vert(&el1.near, &el2.near, @.near);
               call  far_vert(&el1.far,  &el2.far,  @.far);
               call xor_lists(&el1.children, &el2.children,
                              @.children);
               madeof[1]=&el1.id; madeof[2]=&el2.id);
    make (Context current=check_for_duplicate; object=&max_id);
    make (Context current=check_merge_focus;
                  object=&el1.id; using=&el2.id);
};
```

Figure 8.26 This rule is used to merge two faces into a larger face.

CHAPTER 9

COMPLEXITY ISSUES IN BLACKBOARD PROCESSING

In the most general sense, PSEIKI's geometric matching activity can be expressed as the problem of finding subgraph-isomorphisms, a known NP-complete problem [Gar-Joh79]. It is well known that artificial intelligence's use of hueristics can greatly improve the computational efficiency of the solution to a problem solving task; in fact, it has been shown that some heuristics can beat the exponential explosion associated with NP-complete problems [Pea84]. It is hoped that the heuristics encoded into PSEIKI's opportunistic control flow and geometric constraints, when combined with the hierarchical structure of the matching task, will enable PSEIKI to perform matching as scene complexity grows.

There are a number of ways that a system's time and space complexity can be analyzed. If the system's solution to a task can be expressed in a simple, algorithmic fashion, then its complexity often can be calculated theoretically [AhoHop74]. If a system's solution can not be expressed in a way that allows its complexity to be analyzed directly, then the system's major components can be modeled and the model analyzed. Petri net theory [Pet81], one technique for modeling systems, will be explored in this chapter. Particular attention will be focused on stochastic Petri nets, an extension to Petri net theory created by associating an exponentially distributed firing time with each transition in the net [Mol82]. Stochastic Petri nets can be analyzed by mapping the state-space of the net to a Markov-chain and by using concepts from queuing-theory to analyze the system. Currently, stochastic Petri nets can model only small-scale systems because the state-space of a Petri net grows exponentially with the size of the net (hence, so do the nodes in the Markov-chain).

If a system is too complex to be analyzed theoretically or modeled effectively, as is currently the case with blackboard systems, the only resort is to determine empirically the system's computational complexity. In the past, experimental investigations have been used to study how control flow [GarCor87] and data locking [FenLes77] affect blackboard performance. Note, since PSEIKI's hierarchical structure and geometric constraints have been fixed, PSEIKI's computational efficiency can be increased mainly by optimizing its control flow.

9.1. System Modeling with Petri Nets

Petri Net theory is a graph based modeling technique that has proven very powerful for modeling concurrent, synchronous and asynchronous systems. Since their introduction by C. A. Petri in his Ph.D. dissertation [Pet66], Petri nets have been used to model complex systems in many diverse domains. Some of these domains include the modeling of production systems, chemical reactions and legal systems (see [Pet81] for a bibliography of some domains of application). Because Petri nets have been used to model such a wide variety of systems and have been used by researchers with a wide range of backgrounds, they have been formulated in many different ways. The definition and development of Petri nets in this report will follow that found in [Pet81]; the reader is referred there for a more complete introduction to Petri net theory and some typical applications.

Formally, a *Petri net graph* is a directed, bipartite multigraph, $G = (V, A)$. V is the set of vertices, $V = \{v_1, v_2, \cdots, v_s\}$ and A is the set of arcs, $A = \{a_1, a_2, \cdots, a_r\}$ where an arc, a_i from vertex v_j to vertex v_k is expressed as $a_i = (v_j, v_k)$ with $v_j, v_k \in V$. Since the graph is bipartite, the set of vertices, V, can be partitioned into two disjoint parts, $P = \{p_1, p_2, \cdots, p_m\}$ and $T = \{t_1, t_2, \cdots, t_n\}$ such that each arc in A contains exactly one vertex in P and one vertex in T. Using the normal terminology, the set P is called the set of *places* and the set T is called the set of *transitions*.

A *Petri net structure*, C, is a four-tuple $C = (P, T, I, O)$. P and T are places and transitions as described previously. The input and output functions, I and O, respectively, map transitions, t_j, to collections of places. The collection of places $I(t_j)$ and $O(t_j)$ are called the input and output places for transition t_j. The *multiplicity* of the arcs between a transition and one of its input places is equal to the number of arcs from the place to the transition. Likewise, the multiplicity of the arcs between a transition and one

of its output places is equal to the number of arcs from the transition to the place[†]. The *marking* of a Petri net is a mapping, μ, from the set of places to the non-negative integers, N.

$$\mu: P \rightarrow N$$

$\mu(\cdot)$ defines the *state* of the net. During execution of the Petri net, the marking of the net may change; that is, the function $\mu(\cdot)$ may change reflecting the evolving state of the net. The formal definition of a *marked Petri net structure* (hereafter merely called a Petri net) is $M = (P, T, I, O, \mu)$ with the previously defined components.

Although Petri nets are defined in abstract, graph-theoretic terms, it is often helpful to draw the marked Petri net graph. When drawing Petri nets, a *bar* | represents a transition and a *circle* O represents a place. Tokens, drawn as small dots • in a given place, p_j, are used to represent the value of $\mu(p_j)$. An input place of a transition is indicated by an *arrow* from the place to the transition. Conversely, an output place of a transition is indicated by an arrow from the transition to the place. Fig. 9.1 shows an example of a simple Petri net; Fig. 9.2 shows its associated graph.

$$P = \{p_1, p_2, p_3, p_4, p_5\}$$
$$T = \{t_1, t_2, t_3, t_4\}$$

$I(t_1) = \{p_1\}$ $\qquad$ $O(t_1) = \{p_2, p_3, p_4, p_4\}$

$I(t_2) = \{p_2, p_3, p_4\}$ $\qquad$ $O(t_2) = \{p_2\}$

$I(t_3) = \{p_4, p_4\}$ $\qquad$ $O(t_3) = \{p_5\}$

$I(t_4) = \{p_5\}$ $\qquad$ $O(t_4) = \{p_3, p_4\}$

$\mu(p_1) = 1;$ $\quad \mu(p_2) = 0;$ $\quad \mu(p_3) = 0;$ $\quad \mu(p_4) = 2;$ $\quad \mu(p_5) = 1$

Figure 9.1 This figure shows an example of a simple Petri net.

[†] Note that the input and output multiplicities between a transition and a place need not be equal if the place is both an input place and an output place for the transition. The multiplicities will differ if the number of arcs from the place to the transition is different from the number of arcs from the transition to the place.

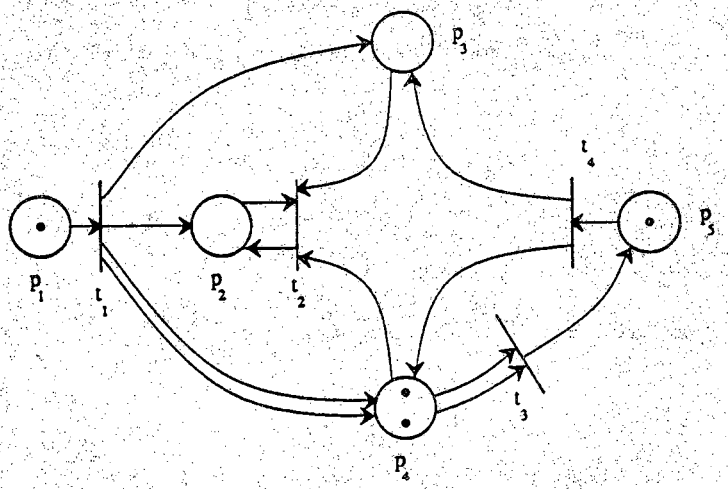
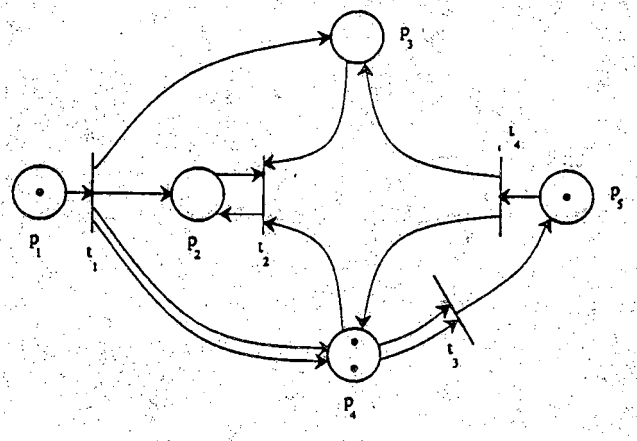


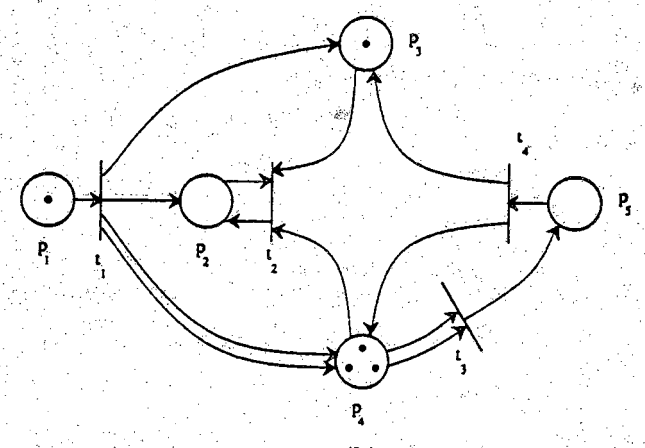
Figure 9.2 This is the marked Petri net graph for the Petri net given in Fig. 9.1.

A transition is said to be *enabled* when the number of tokens in each of the transition's input places is greater than or equal to the multiplicity of the arcs between the transition and that input place. For example, if there are two arcs from an input place to a transition, then the transition will not be enabled until there are at least two tokens in that input place. An enabled transition is *fired* by removing tokens from the transition's input places and adding tokens to the transition's output places. The number of tokens removed from or added to the transition's input places or output places, respectively, is equal to the multiplicity of the arcs between the transition and the places. If more than one transition is enabled at any time, then the transition that is fired is picked at random. In general, the state of the net will change when a transition fires. Thus some transitions that previously were enabled may no longer be enabled and some new transitions may become enabled. The process of successively firing enabled transitions is called *executing* the Petri net. When there are no enabled transitions, the execution of the Petri net *halts*. Fig. 9.3 shows the execution of the Petri net shown in Fig. 9.1. Panel (a) in this figure shows the net's initial marking. Panel (b) shows the net's marking after t_4 fires and panel (c) shows the net's marking after t_1 fires.

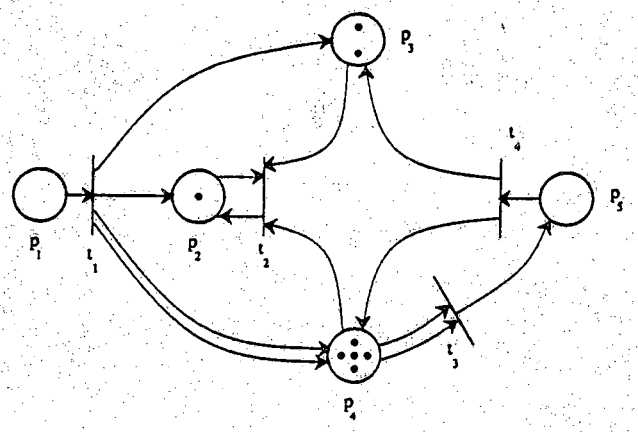
A marking of a Petri net is said to be *reachable* from another marking if there is a sequence of transition firings that transforms the state of the net from the initial marking



(a)



(b)



(c)

Figure 9.3 This figure shows the execution of the Petri net from Fig. 9.1.

to the desired marking. The *reachability set* of a marking is defined to be the set of all states reachable from the initial marking. Note that the reachability set of a Petri net is dependent on the original marking. Also note that the reachability set of a Petri net will grow exponentially with the number of places, transitions, and tokens present in the net. Both of these affects limit the usefulness of Petri nets in the modeling of blackboard systems.

Fig. 9.4 is a simple example of a Petri net that could be used to model PSEIKI's flow of control. The places in this net correspond with the blackboard scheduler and knowledge sources. The token represents the locus of processing in the system; a process is considered active when its corresponding place contains the token. Notice that the configuration of the net forces the control of the system to return to the scheduler between each knowledge source activation. The net can be extended to model concurrent blackboards by adding a token for each processing thread. Obviously, the model shown here is over-simplified and cannot be used in any realistic analysis.

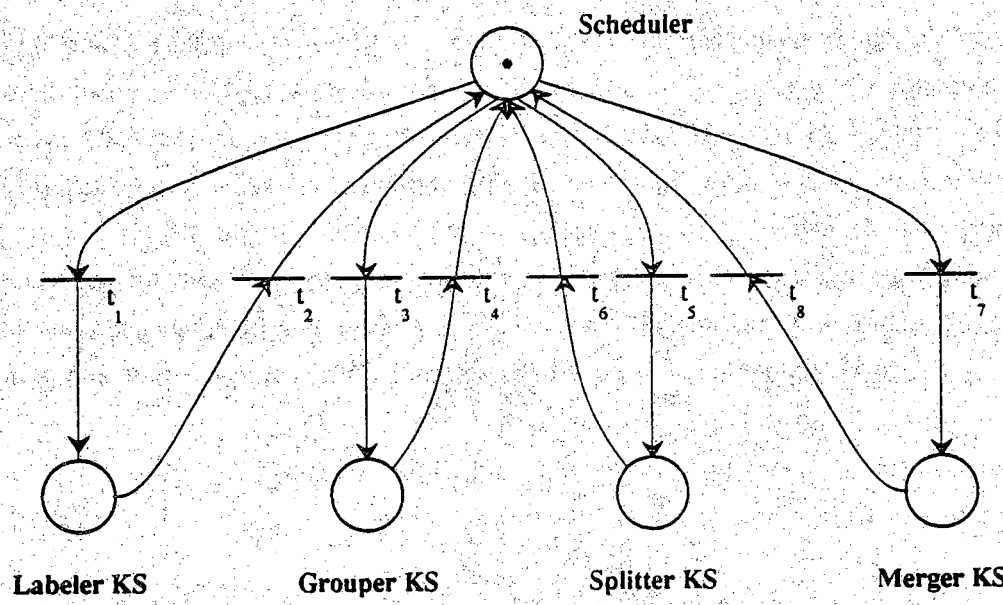


Figure 9.4 This figure shows a simple Petri net that can be used to model PSEIKI's control flow.

Petri net theory has been extended in a number of ways to make it a more powerful modeling tool. Stochastic Petri Nets, an extension first proposed by Molloy [Mol82], are created by associating an exponentially distributed firing time with each transition. The firing time of a transition specifies the average amount of time that the transition takes to fire. Thus the transitions in a stochastic Petri net will fire a random amount of time after they become enabled (unless another transition fires first and disables the first transition). If another transition fires but does not disable the first transition, then the timing of the first transition does not change (the first transition does not have to be "reset" because of the memoryless property of the exponential distribution).

A stochastic Petri net is formally defined as $S = (P, T, I, O, \mu, \lambda)$ where λ is the mapping from the transitions to the real numbers that defines the mean firing time of the exponentially distributed random processes. The rest of the components of S have been defined previously. Note that the transitions' firing rates are specified completely by λ because an exponential distribution is specified completely by its mean value.

Stochastic Petri nets are useful tools for analyzing complex systems because they are isomorphic with homogeneous Markov processes but have all the expressive capabilities of the original Petri nets [Mol81]. The isomorphic properties of a stochastic Petri net and a Markov process can be seen with the help of the following example. In this example, the simple Petri net shown in Fig. 9.5 will be converted into an equivalent Markov chain. The first step in the conversion process is the determination of the reachability set of the net given an initial marking. The reachability set of the example Petri net is given in table 9.1. Each row in this table represent a distinct state of the net. The entries in the table represent the number of tokens in a place for a given state. If the mean firing times of the transitions in the stochastic Petri net shown are $\lambda_1 = 2$, $\lambda_2 = 1$, $\lambda_3 = 1$, $\lambda_4 = 3$, $\lambda_5 = 2$, then the following procedure can be used to map the state-space of the net to a Markov chain. A state in the chain is created for every distinct marking in the net. A state-transition is created between two states in the chain if the firing of a single transition in the Petri net will transform the marking of the net from the first state to the second. The mean transition time of the state-transition is set to the mean firing time of the transition that must fire to transform the state of the net from the first state to the second state. For example, marking μ_2 will be transformed into marking μ_4 if transition t_3 fires; thus, in the Markov chain, there is a state-transition from state μ_2 to μ_4 with an average transition time of 1 second, the mean firing time of transition t_3. Fig. 9.6 shows a Markov-chain that is isomorphic to the net shown in Fig. 9.5. In this figure, the mean

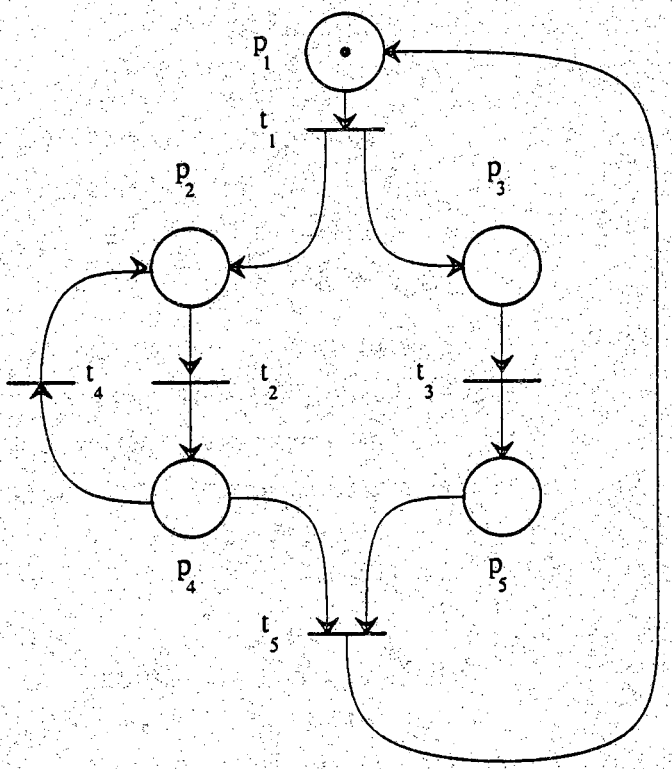


Figure 9.5 This figure shows a simple Petri net that is used in the textual explanation of the isomorphism between stochastic Petri nets and Markov processes.

Table 9.1 This table shows the reachability set of the Petri Net shown in Fig. 9.5.

	p_1	p_2	p_3	p_4	p_5
μ_1	1	0	0	0	0
μ_2	0	1	1	0	0
μ_3	0	0	1	1	0
μ_4	0	1	0	0	1
μ_5	0	0	0	1	1

transition times between states of the chain are indicated by the numbers shown above the state-transitions. The numbers shown below the state-transitions are the *transition-probabilities* of the chain.

Once an equivalent Markov chain is constructed from a stochastic Petri net, classical queuing theory techniques [Tri82] may be used to determine the performance of the system by analyzing the chain. For example, the throughput of a system can be estimated by determining the average amount of time that the system needs to transform from a starting state to an ending state and then reset back to the starting state.

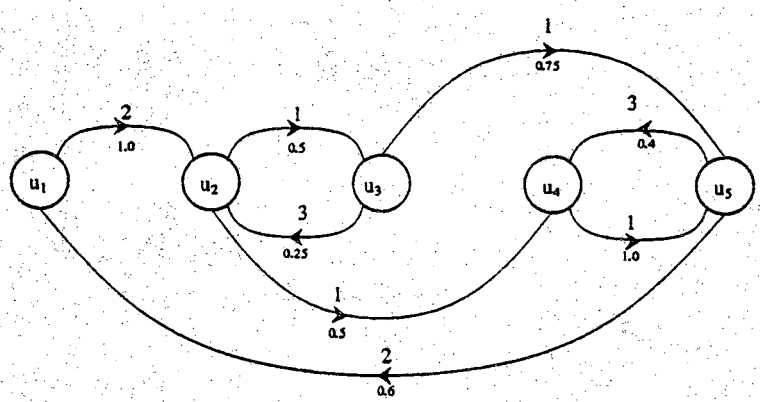


Figure 9.6 This figure shows the Markov equivalent to the stochastic Petri net shown in Fig. 9.5.

Queuing theory techniques also can be used to determine the the *steady-state marking probabilities* of the system (the probability that the net will have a particular marking at a given time) by determining the equivalent chain's *limiting state probabilities*. By finding the limiting state probabilities of the Markov-chain in Fig. 9.6, the steady-state marking probabilities of the net in Fig. 9.5 can be shown to be

$$P[\mu_1] = 0.1163$$

$$P[\mu_2] = 0.1860$$

$$P[\mu_3] = 0.0465$$

$$P[\mu_4] = 0.5349$$

$$P[\mu_5] = 0.1163$$

In their current state of development, stochastic Petri nets have a number of drawbacks that limit their use for modeling blackboard systems. First, the reachability set of the net depends on the initial marking. Thus if tokens are used to represent data elements on the blackboard or other problem dependent information, then a new analysis is needed for each problem instantiation. Second, the current formulation of stochastic Petri Nets requires that **every** transition have an exponentially distributed firing time. When modeling complex systems, such as blackboards, it may be neccessary to model transitions that fire immediately on enabling, require a fixed amount of time to fire, or fire in an amount of time that is a function of the net marking. In addition to these limitations, a final drawback prohibits the use of stochastic Petri Nets for modeling large-scale systems. In general, the size of a Petri net's reachability set will grow exponentially as the number of tokens, places, or transitions in the net increases. Since most queuing theory techniques require the determination of the eigenvalues and eigenvectors of an $N \times N$ matrix when solving a Markov-chain with N states; the problem quickly becomes intractable as its size increases. Although stochastic Petri nets currently cannot model systems as complex as blackboards, most researchers are optimistic about the prospect of extending them to handle such large-scale systems. See [RamHo80], [MarCon84], [Zub85], [DugBob85] for some recent work on extended stochastic Petri nets.

CHAPTER 10

MOBILE ROBOT SELF-LOCATION WITH THE PSEIKI SYSTEM

In the mobile robotic context, knowledge of the scene expected to be visible to a robot-mounted camera is a powerful tool for updating estimates of the robot's position and orientation as the robot travels through a building. The PSEIKI system has been used successfully for autonomous navigation of a mobile robot in indoor environments. In these experiments, PETER†, the mobile robot at Purdue's Robot Vision Lab, used vision data to navigate through building corridors. Fig. 10.1 shows, from two vantage points, the building corridors in which the experiments were conducted. This figure depicts the hallways in the lab area of our building, with doors, bulletin boards, etc., at various locations along the walls. The floor is made of semi-gloss tiles; these are a source of glare in camera images.

A photograph of the mobile robot PETER is shown in Fig. 10.2; Fig. 10.3 shows a diagram of the robot with its main components labeled. As can be seen in the diagram, the robot is equipped with a number of sensors. Two cameras are mounted near the top of the robot to enable it to navigate using stereometric vision. Only a single camera was used in the experiment described here. This camera was aimed downward such that the robot sees only about fifty feet down the corridor; this makes for a near-sighted robot [KakRob87]. The image data from the camera is transmitted via a video RF link to a host SUN 3 computer where it is digitized. Another RF link is used to send commands to the robot and to query its status. A ring of SONAR sensors is mounted on the robot for real-time collision avoidance capabilities. The robot is equipped with a set of encoders

† A Programmable Engine for Terrain Exploration Research

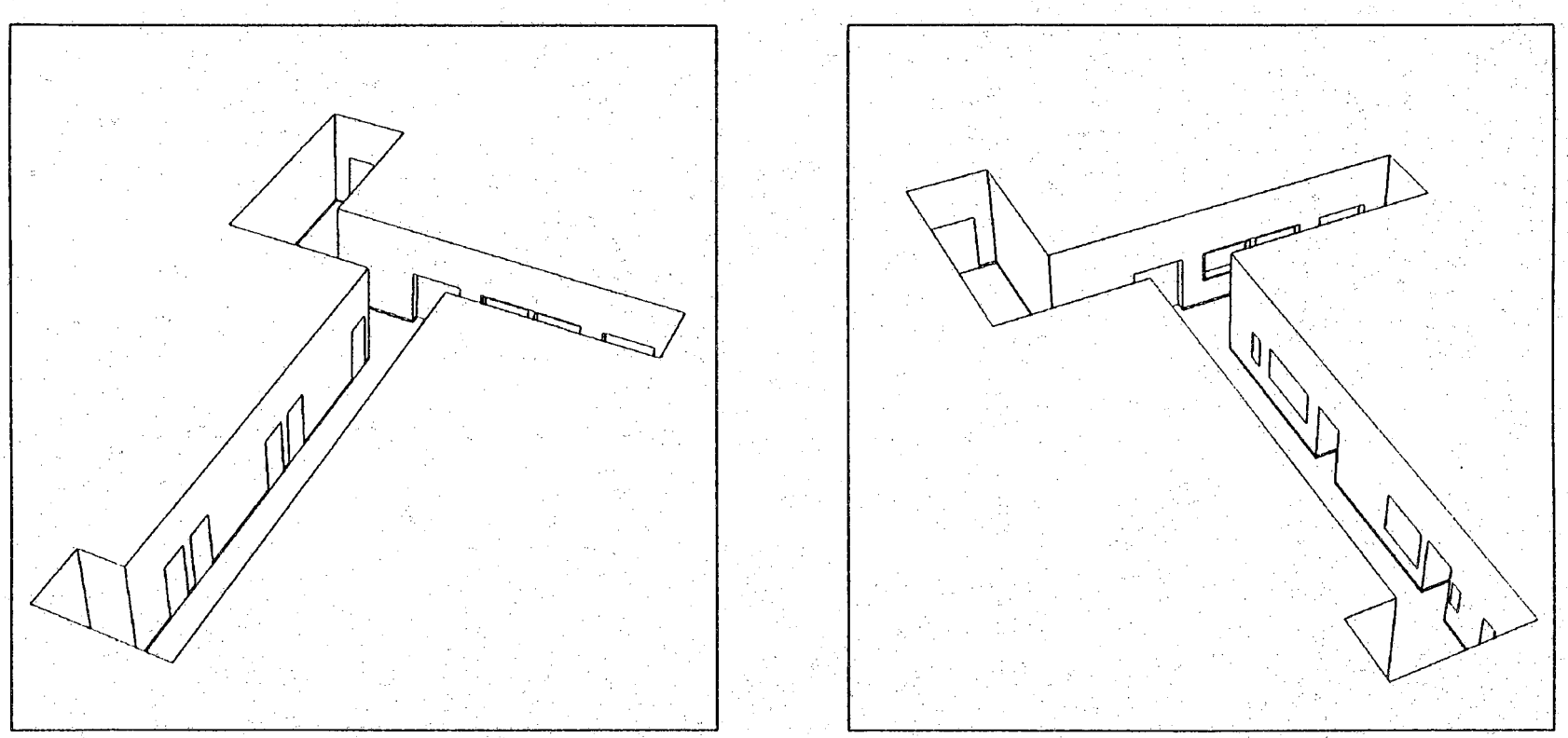


Figure 10.1 This figure shows the building corridors used in the mobile robot self-location experiments.

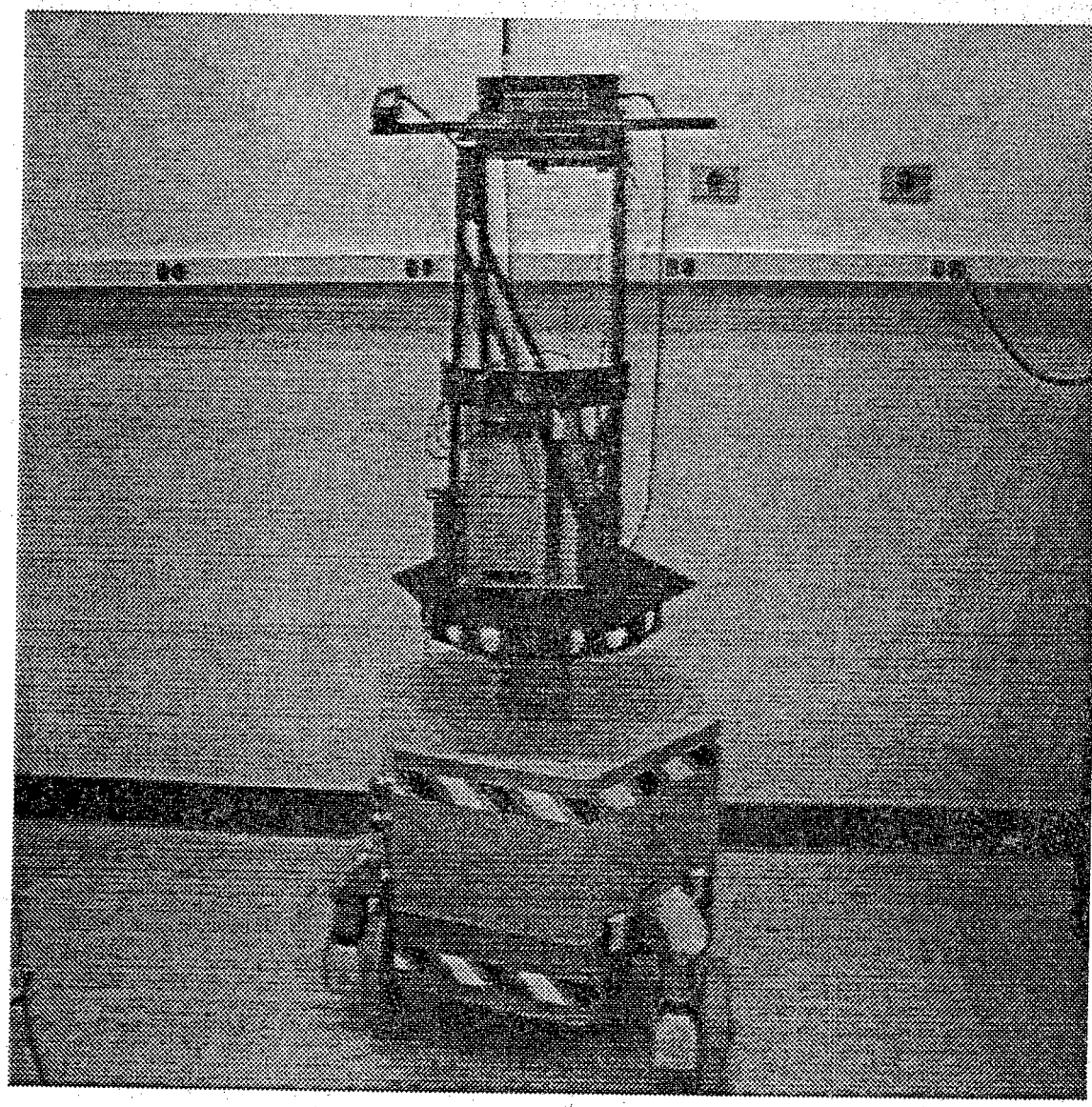


Figure 10.2 This figure shows a photograph of PETER, the mobile robot at Purdue's Robot Vision Lab.

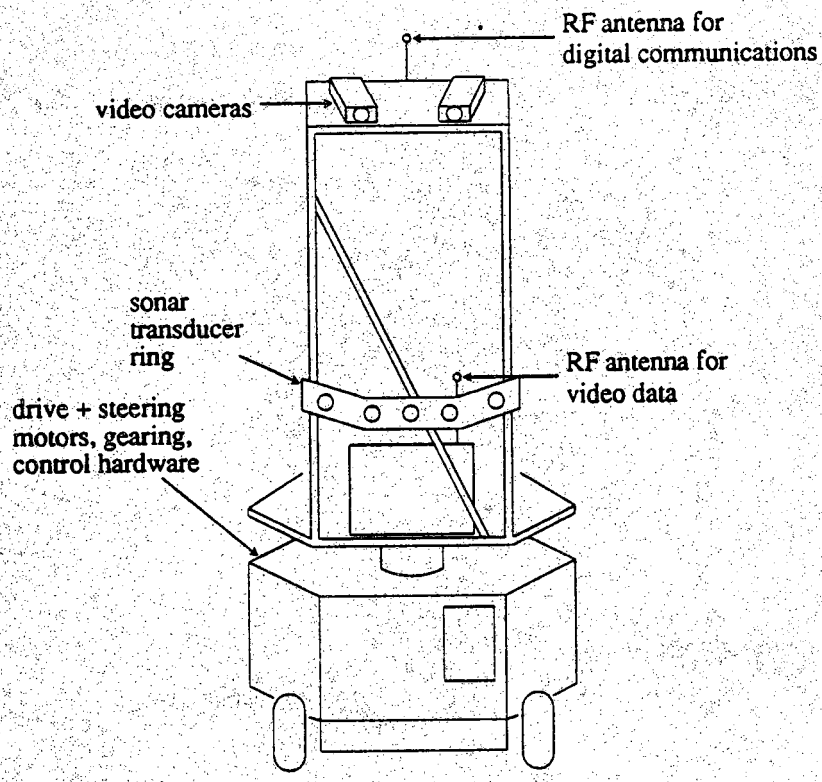


Figure 10.3 This figure shows a diagram of the robot with its main components labeled.

mounted on its steering motor and its wheels; these encoders give the robot the ability to determine its approximate position and orientation based on dead-reckoning. A small on-board 68000 based computer is used as a supervisor. A more complete description of the mobile robot, its architecture and capabilities can be found in [LopKak89].

In this experiment, the robot's task is to travel between two specified points in the building. As mentioned previously, the robot is equipped with a set of encoders to allow it to perform "inertial" navigation. However, for a number of reasons, the exact position and orientation of the robot is never known with certainty. To give the reader an idea of the quality of odometry of the robot, in many instances, a commanded turn of 45° introduced an orientation uncertainty of 2°. Straight-line motions had a 10% uncertainty in the distance traveled. Even worse, due to uneven weight distribution in the base of the robot and differences in the diameters of the wheels, a command to travel straight in a certain direction usually resulted in motion along a circular arc resulting in a motion that could be up to 15° off from the commanded direction. It was not possible to construct a usable model of this uncertainty as the uncertainties depended strongly on factors such as the starting orientation of the robot, whether or not the floor had been waxed recently, etc. The accumulation of these errors as the robot travels further from its point of origin results in decreased certainty in its position and orientation. Without the aid of sensory feedback, the uncertainty grows to a point that it is impossible to guarantee that the robot will not bump into a wall, etc.

Therefore, it is not possible to have the robot travel by dead-reckoning alone; sensors must be used to update the robot's hypothesized position and orientation as it travels through the hallways. To keep the uncertainty in the robot's position and orientation at a reasonable level, PSEIKI is used to interpret vision information and improve the estimate of the robot's position and orientation. If the mobile robot's position and orientation are known exactly, then it is possible to render an image corresponding to what the camera mounted on the robot should see. If, due to odometry errors, there is an error in the hypothesized estimate of the position and the orientation of the robot, then there will be a discrepancy between what the camera is expected to see and what actually is seen. It is important to realize that this discrepancy will not be a simple translation of the expectation scene with respect to the perceived image due to the three dimensional geometry involved. After PSEIKI has completed the matching of elements in the image data with expected scene elements, a self-location procedure to be described later is used on the most believed image-data/model-data pairs to update the robot's hypothesized location.

Because of the computational costs involved, such exercises in self-location cannot be carried out continuously. Thus, the robot's trip is divided into a number of short moves; the self-location procedure is performed at the end of each move. The positions where self-location is performed are called way-points. How far the robot can go before it must self-locate is a function of the quality of the odometry and the maximum misregistration that PSEIKI can tolerate for the purpose of "matching" the perceived image with the expected scene.

Fig. 10.4 schematically shows the building corridors in which the experiments were held. In many of the trials, the task of the robot was to navigate autonomously from point A to point B. The total distance between those two points is approximately 40 meters. To cope with the uncertainties introduced by poor odometry, a conservative 6 meters was chosen as the longest distance the robot was allowed to travel without updating its location and orientation through PSEIKI. In these trials, the initial position and orientation of the robot was known to within 10 cm and 5°, respectively. The region-based low-level image preprocessor described in chapter 3 was used to convert raw image data into symbolic form. The solid-model based expected scene generator from chapter 4 was used to provide the model data.

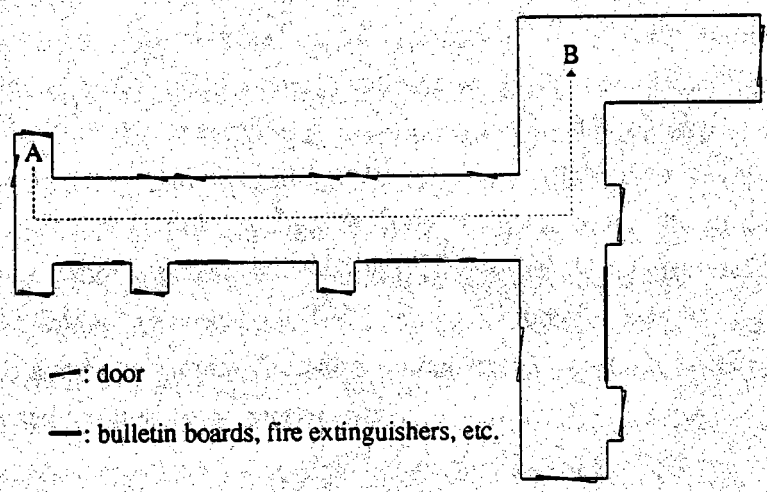


Figure 10.4 This figure shows the path the robot took in many of the trials.

A solid model of the building's corridors was generated off-line and was used to generate the robot's expected scene. This model contained representations for all major structures in the corridor, such as alcoves, doorways, etc. Because the expected scene generator arbitrarily assigns labels to the model information presented to PSEIKI, the labels of all edges were changed by hand to agree with their labels as they were stored in the TWIN database. This tedious process was required for each expected scene in the experiment.

Fig. 10.5 shows a block diagram of the system used in the experiments. As can be seen in this figure, an estimate of the robot's initial position and orientation is input to the top-level system at the start of an experiment. The expected scene generator uses this estimate, camera calibration information and the solid model of the hallway to generate a symbolic description of the expected scene. Vision data is generated by applying the region-based segmenter to a digitized image of the scene transmitted by the mobile robot. After the input data is generated, PSEIKI identifies expected-scene elements in the observed image and outputs the elements from the most believed scene interpretation (the children of the data-panel scene element with the largest belief). The self-location procedure is then applied to the output data to update the hypothesized location of the robot. If the updated position is less than 10 cm. from the goal position, then the experiment is successfully terminated; otherwise, the location of the next way-point is calculated. If the goal point or the next turn point in the path plan is less than 6 meters away from the position of the robot, then that point will be used as the way-point. Otherwise, the way-point is defined to be the point 6 meters from the present position of the robot in the direction of the next turn point (or the final goal point). Commands are then transmitted to the robot to move to the next way-point. The location of the next way-point is also presented to the expected scene generator for the next self-location cycle.

The list of matches and the associated belief values output by PSEIKI are used to update the robot's hypothesized position. By using the matches and geometric information stored in the TWIN database, it is possible to determine, in the world coordinate frame, the equations of the 3D lines that produced the edges in the observed image. With this information and knowledge of the camera calibration characteristics, the position and orientation of the robot can be determined. Only the edges with a belief value exceeding some threshold, usually 0.5, are used in the self-location procedure.

The actual calculation of the robot's location is carried out by keeping track of two coordinate systems: the world coordinate system, represented by $\mathbf{W}^3$, in which the hallways are modeled, and the robot coordinate system, represented by $\mathbf{R}^3$, which translates

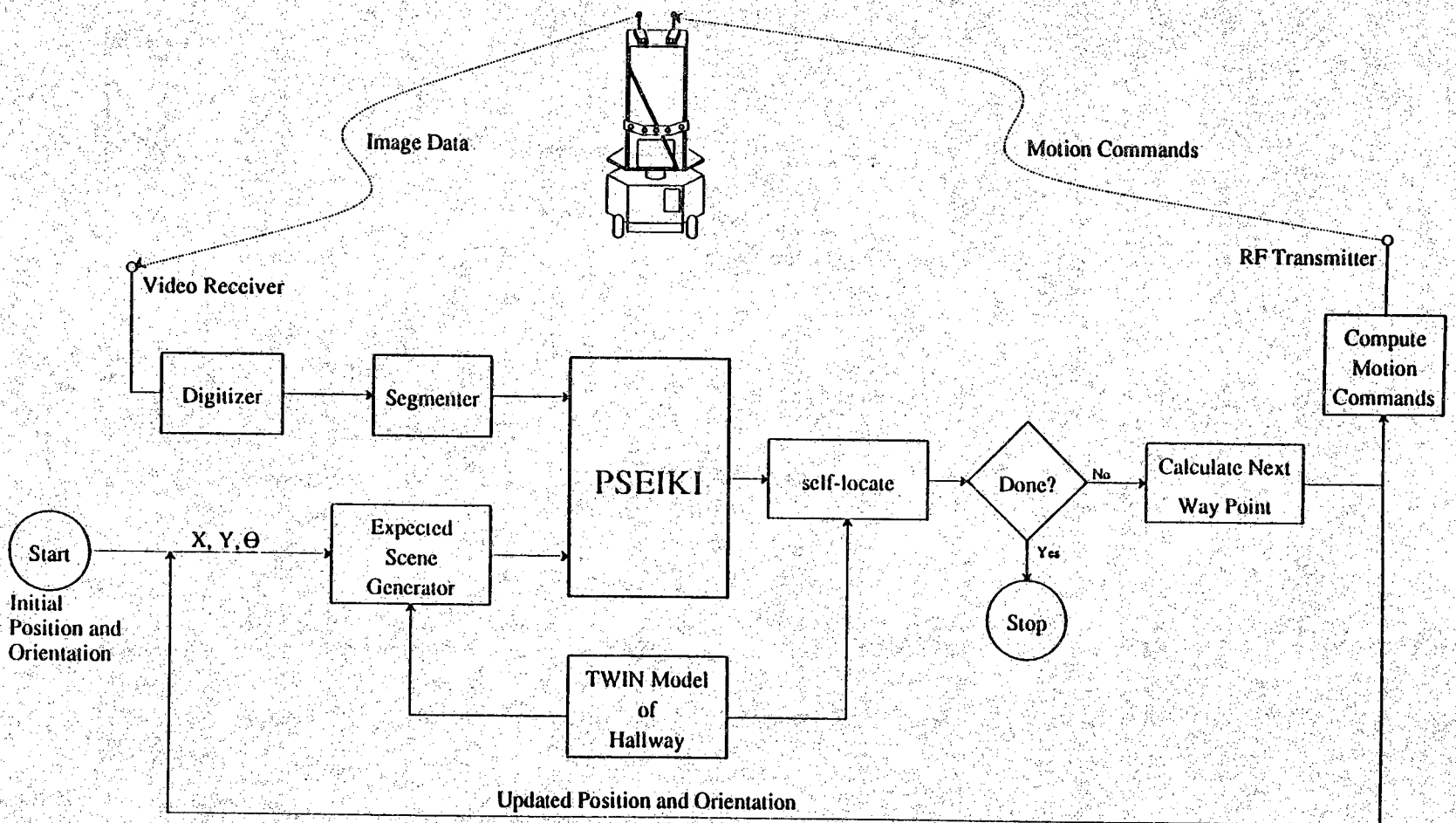


Figure 10.5 This figure shows a block diagram of the mobile robot self-location experiment using PSEIKI.

and turns with the motions of the robot. The camera is calibrated in $\mathbf{R}^3$; the calibration parameters make it possible to calculate the line of sight in $\mathbf{R}^3$ to any pixel in the image. The problem of robot self-location is to compute the position and the orientation of $\mathbf{R}^3$ with respect to $\mathbf{W}^3$. In these experiments, it has been assumed that the origin of $\mathbf{R}^3$ always stays in the xy-plane of $\mathbf{W}^3$ and that their z-axes are parallel and designate the vertical. The orientation of $\mathbf{R}^3$ is defined to be the angular rotation of the xy-plane of $\mathbf{R}^3$ with respect to the xy-plane of $\mathbf{W}^3$.

The procedure used to solve the self-location problem, given the edge correspondences between image and model data, is described in [LopKak89]. To summarize the procedure described there, the problem of robot self-location can be decomposed into two sub-problems: the problem of finding the orientation of $\mathbf{R}^3$ and the problem of the finding the coordinates of the origin of $\mathbf{R}^3$ in the xy-plane of $\mathbf{W}^3$. As shown in [LopKak89], the orientation of the robot can be found from a single pair of image/model edges provided that the model edge is not vertical. Using this procedure, the orientation of the robot can be determined to within a multiple of 180° with a single image/model edge pair. It is easy to choose the correct orientation by comparing the two possibilities with the reading of the corresponding encoder. If more than one image/model edge pair is found, a weighted average is taken of the orientation estimates produced by the different pairs; the weight for each pair is proportional to the belief value associated with the edges. The orientation of $\mathbf{R}^3$ derived through dead-reckoning also is averaged into the updated orientation estimate. This estimate is averaged into the updated hypothesis for a number of reasons. First, this estimate can be fairly accurate in many cases. Second, it is possible that no suitable non-vertical edge will be found in the observed image; in this case, the dead-reckoning estimate will be the only contributor to new orientation estimate.

Once the orientation of $\mathbf{R}^3$ is known, two different approaches are used simultaneously to compute the coordinates of the origin of $\mathbf{R}^3$ in $\mathbf{W}^3$. The first approach relies on the fact that it is possible to compute the perpendicular distance of the origin of $\mathbf{R}^3$ to a model edge if that edge is horizontal and is the label element of an edge in the image data. Therefore, if PSEIKI can find matches for two non-parallel horizontal lines in the model, the world coordinates of the origin of $\mathbf{R}^3$ are computed easily. The second approach computes the location of the origin of $\mathbf{R}^3$ given any two image/model edge pairs if the model edges are not parallel. Again, the results produced by both these approaches, for all possible pairs of edges satisfying the necessary conditions, are

averaged using weights that depend upon the beliefs associated with the edge pairs. The location of $\mathbf{R}^3$ derived through dead-reckoning also is averaged into updated orientation estimate for the reasons outlined above.

Figs. 10.6-10.10 show examples of images typical of those used in the self-location experiments. Fig 10.6 shows a line drawing of the scene expected to be visible by the robot-mounted camera. This image was generated by rendering the solid-model of the hallways using the calibration parameters of the camera on the robot and the position of the robot as supplied by odometry. This model information was deposited onto all levels of PSEIKI's model panel of the blackboard. Fig 10.7 shows the image data collected by the robot. Notice that there is a significant amount of misregistration between the expected scene and image data. Regions and edges were extracted from this image and input into the vertex, edge and region levels of the data panel of the blackboard; the edge-level data presented to PSEIKI is shown in Fig. 10.8. Fig. 10.9 shows the edges output by PSEIKI at the end of processing; these edges were descendents of the scene-level data element with the greatest belief. The edges in Fig. 10.10 are used to indicate the belief in the individual edges from Fig. 10.9; the darkest edges have the greatest belief.

A simple real-time obstacle avoidance system is used to prevent the robot from colliding with objects as it moves between way-points. The obstacle avoidance system uses five sonar transducers to detect objects not stored in the TWIN model of the hallway. The five transducers are aimed horizontally and cover an arc of approximately 90° (45° on either side of the robot's front centerline). The on-board supervisory computer directs the robot to head toward the next way-point as long as none of the sonar transducers detects an object. However, as soon as one of the sensors detects an object, the supervisory computer will direct the robot to turn away from the detected obstacle and proceed until the obstacle is no longer detected. When the obstacle is no longer detected, the robot will turn back toward the way-point and continue onward. PSEIKI deals with objects visible in the image but not represented in the TWIN model (such as stationary obstacles, people, etc.) by assigning low belief to the unknown objects and matching, with high belief, only those objects stored in the model. By matching these known objects with high belief, PSEIKI provides enough matches to the self-location system to update the robot's position even with incomplete expected/detected matches.

The task of planning the robot's path through the corridors is trivial. Besides being represented as a TWIN solid, the building's corridors also are represented in a graph

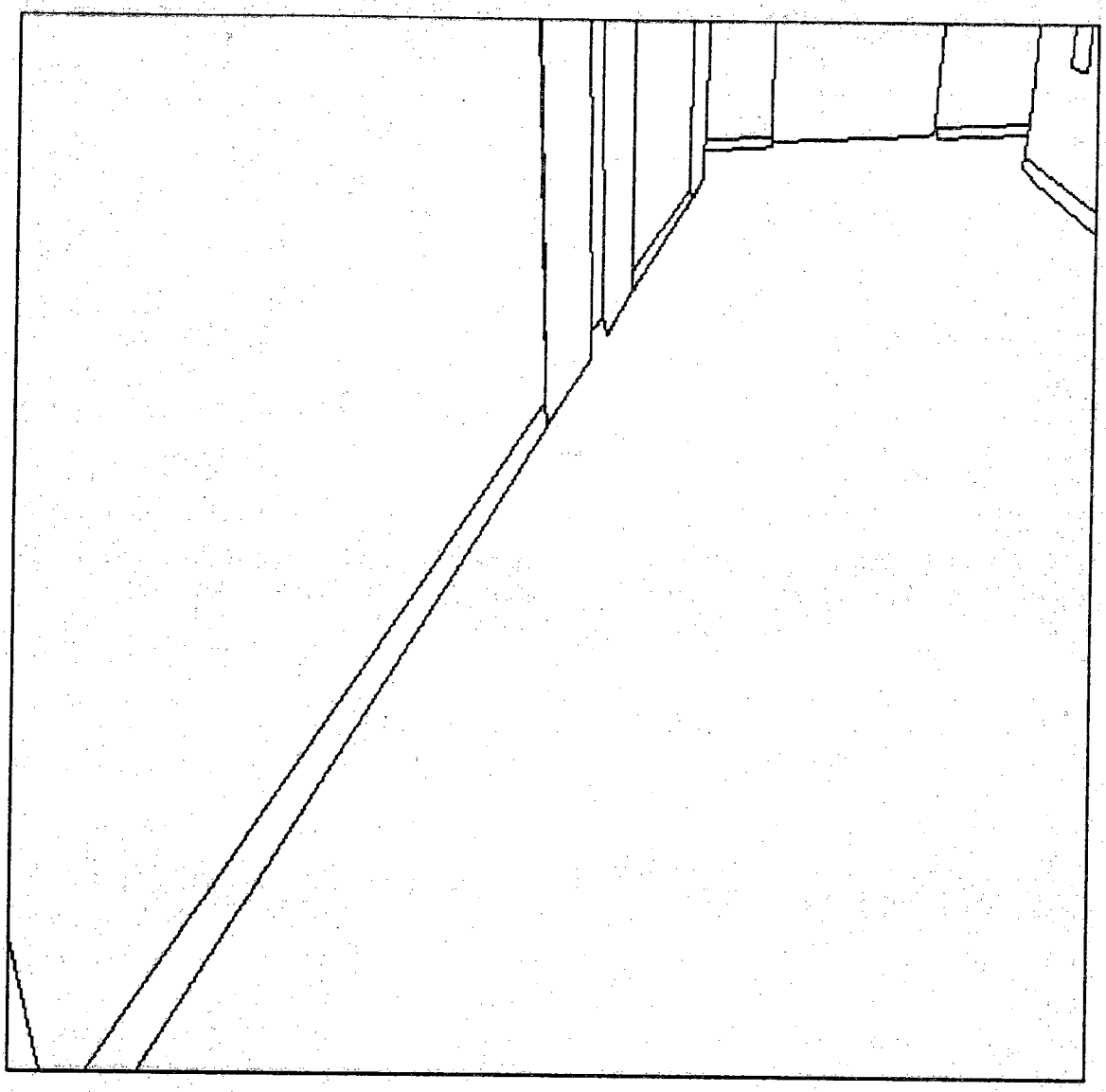


Figure 10.6 This figure shows an example of a typical expected scene from the mobile robot self-location experiment. It depicts a line drawing of the edges expected to be visible to the robot at its hypothesized location and orientation.

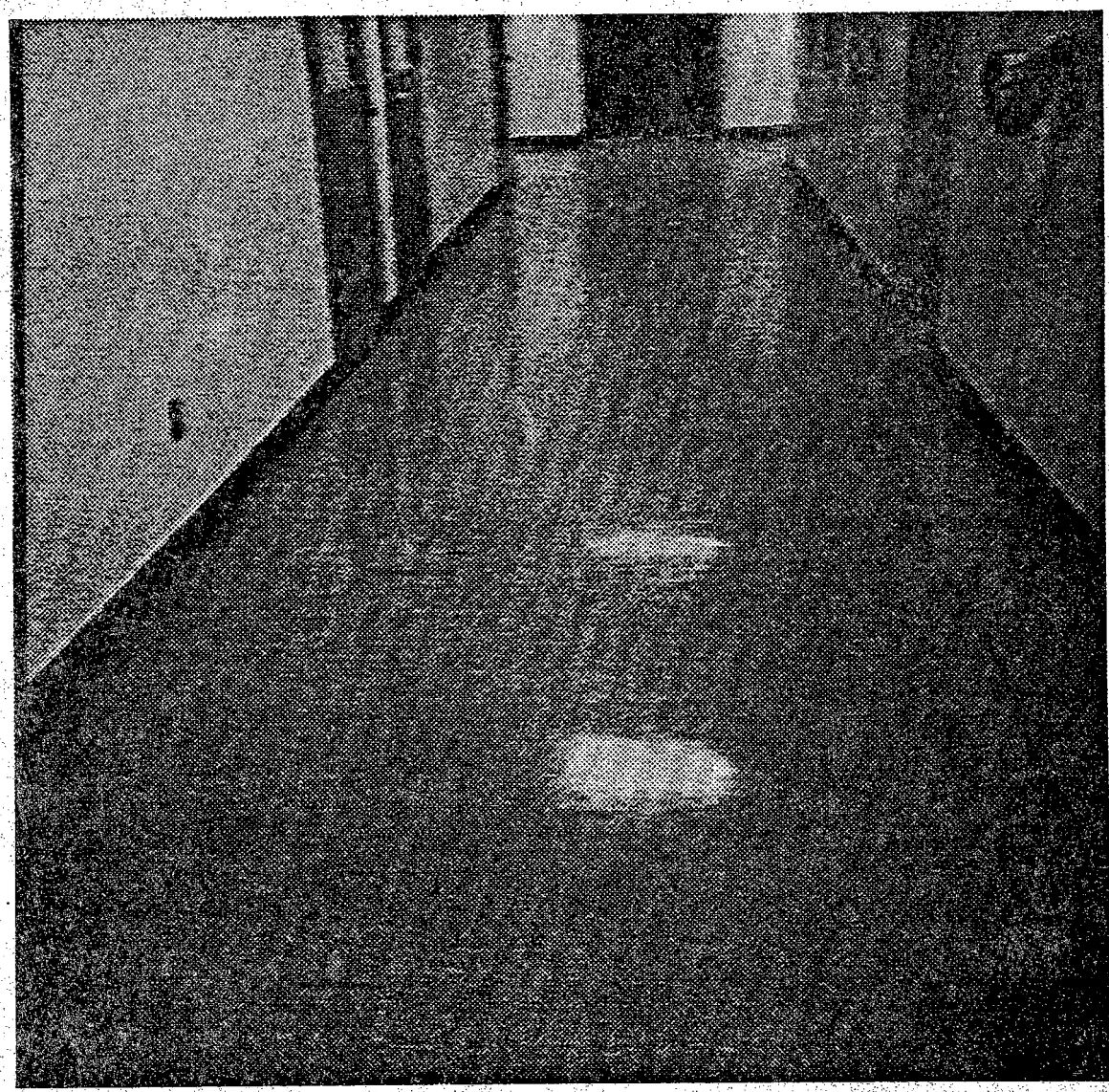


Figure 10.7 This figure shows the scene actually observed by the robot mounted camera. Note the large amount of misregistration between this image and the expected scene shown in Fig. 10.6.

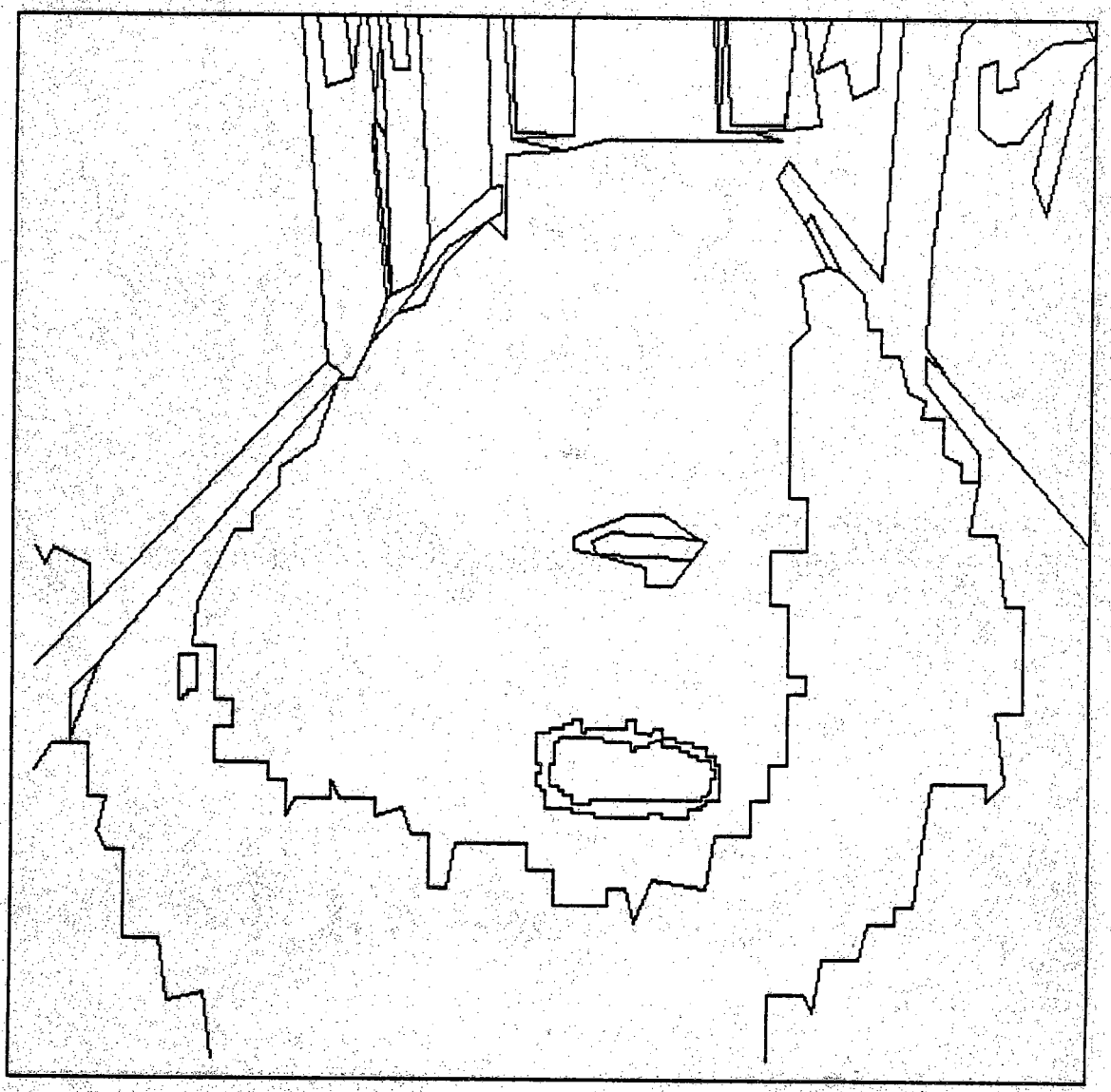


Figure 10.8 This figure shows the edges produced by the region-based preprocessor. The edges denote the borders between the regions found by the preprocessor.

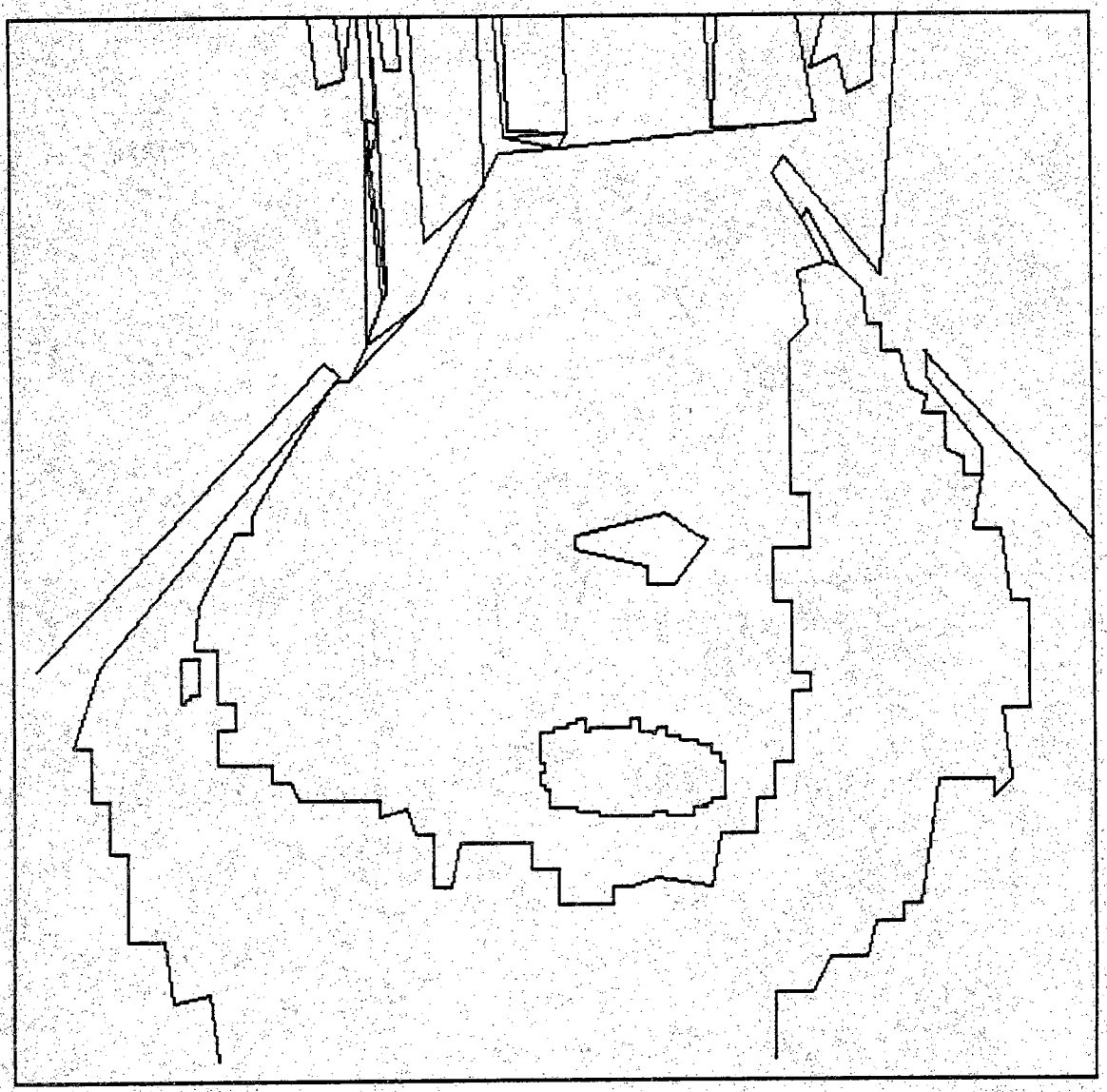


Figure 10.9 This figure shows the edges output by PSEIKI at the end of processing; these edges were descendents of the scene-level data element with the greatest belief.

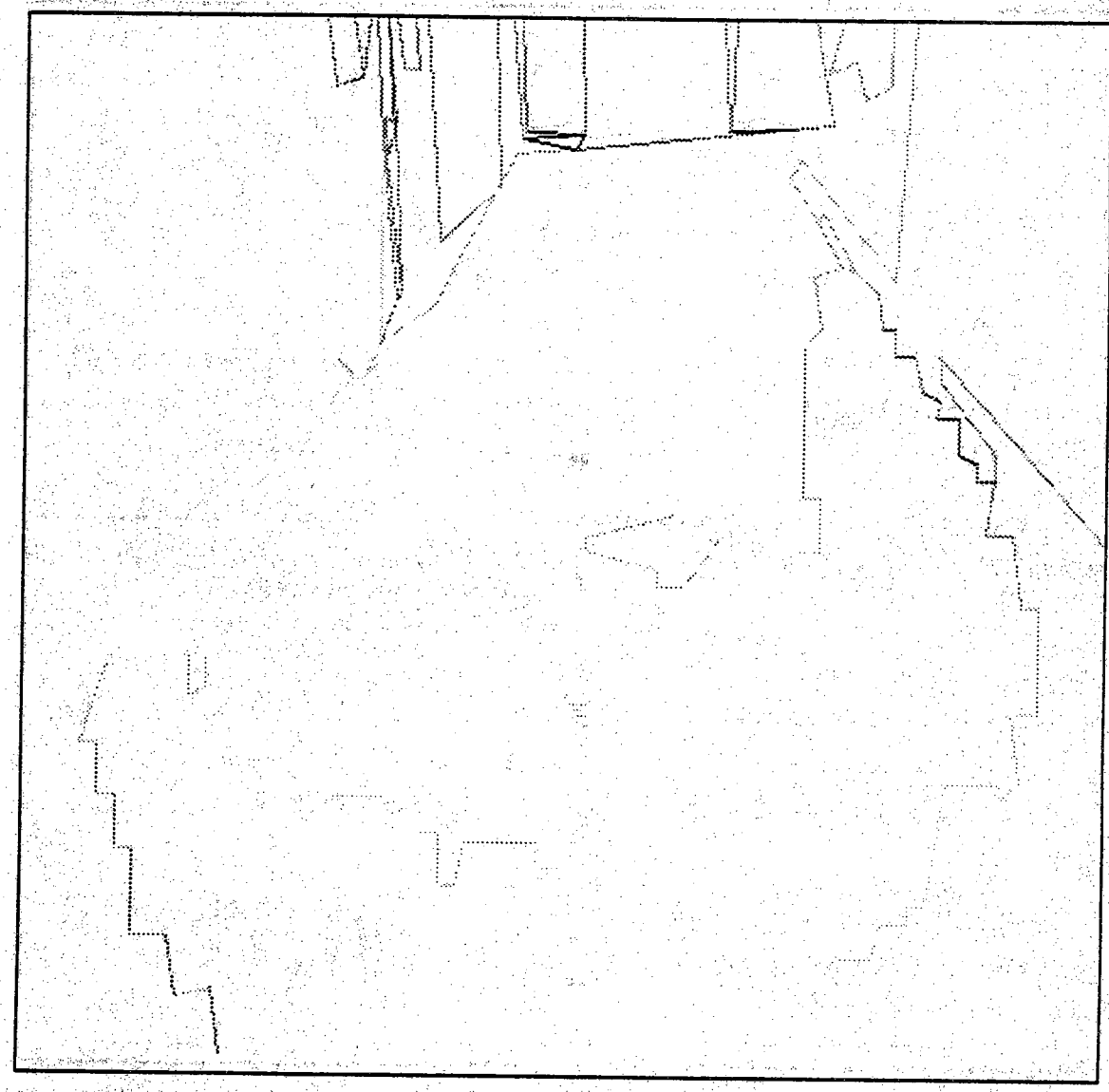


Figure 10.10 This figure indicates the belief in the edges from Fig. 10.9. The darkest edges have the greatest belief.

data-structure. The links in this graph represent straight sections of hallway, and the nodes represent the hallway's junctions and bends. An (x, y) coordinate pair is associated with each node in the graph indicating the location of the junction or the bend in world coordinates. If both the robot's initial position and goal position are at a junction or bend in the hallway, then an A^* best-first search [Pea84] on the hallway graph is performed to find the shortest path from start to goal. If either the initial position or the goal position is not located at a node position, because it is in the middle of a corridor, then the link representing the corridor is split in two and a node is added to the graph to represent the robot's position in the hallway. After the graph has been modified to include nodes for both the initial and goal positions, the best-first search is performed.

At this time, we assume that the robot will never get "lost" in the building (e.g. by making a wrong turn down a corridor). In the future, however, it may be possible to determine that the robot is lost by noting an exceptionally low belief value in PSEIKI's scene interpretation. Once such an error is discovered, a small number of possible positions for the robot may be determined by noting where the robot may have made the error in navigation. PSEIKI could then be used to determine which one of the possibilities is most-likely the robot's true position. Then a new path to the goal position could be planned using the robot's hypothesized position as a new initial state.

CHAPTER 11

CONCLUSIONS AND RECOMMENDATIONS FOR FUTURE WORK

In this document, we have described PSEIKI, a system that is capable of forming an interpretation of an image given knowledge of the expected scene. This system opportunistically matches elements in the image data with elements in the expected scene at a number of levels of abstraction. High level image data constructs are built using cues taken from the expected scene. Belief values are attached to the matches and are updated based on the extent to which geometric relationships between elements in the expected scene are met by elements in the image data. An efficient implementation of Dempster's rule was developed to deal with the tremendous amount of information present in most images. The system has successfully been used to aid in the autonomous navigation of the mobile robot at Purdue's Robot Vision Lab. Although PSEIKI has proven to be an effective tool for expectation-driven image interpretation, there are a large number of extensions that could be used to enhance PSEIKI's utility.

PSEIKI's greatest limitation lies in the method that it currently uses to generate its expected scenes. Although this limitation is not as theoretically interesting as many others, most of the other extensions cannot be addressed until the expected scene generator's limitations are overcome. Most of the expected scene generator's limitations stem from the render/segment process described in chapter 4. First, the expected scene generator assumes that all faces (but the background) should be grouped into a single object. If more than one object is present in the scene, then the upper-level information must be hand corrected. Secondly, the labels attached to the symbolic elements produced by the expected scene generator are generated randomly during the image segmentation phase and do not correspond to the data stored in the TWIN solid model. These correspondences are necessary for finding the relative locations of the camera and the scene objects

via triangulation. Currently the user is required to change the labels assigned to the elements to reestablish the necessary correspondences between the solid model and the symbolic description presented to PSEIKI. If these two limitations were removed, then no human interaction would be required in the expected scene generation process. Thirdly, the expected scene generator does not supply the values of the expected scene elements (such as the expected strength of an edge or expected greyscale of a surface). Currently, it is not possible to store this information in a TWIN solid model. Although it would be possible to extend the TWIN data structure to enable it to store the value information, the information still would be lost during the rendering process because the rendered image contains the surfaces' ID numbers and not their expected values. Finally, PSEIKI's expected scene generator currently specifies the vertex locations in terms of the image plane. Thus, all 3D geometric information is lost in the conversion process. An improved method would retain the 3D information by specifying the vertex locations in the world coordinate frame (given the camera calibration information, PSEIKI would be able to project the data onto the image plane if necessary). Most of these limitations can be overcome by removing the intermediate rendering step and converting the information in the solid models directly into symbolic form.

A fundamental limitation of the expected scene generator which cannot be overcome by removing the intermediate rendering step is its requirement that the objects be represented in polyhedral form. However, if a new system without this limitation was used to generate PSEIKI's expected scenes, then PSEIKI could be extended to handle these more general scene descriptions. For example, if an expected scene generator which contained representations for cylindrical and spherical surfaces and elliptical arcs was developed, then it may be possible to extend PSEIKI to work with these new types of elements. Extending PSEIKI to handle the new data types would require new data structures and evidence generation metrics.

If the expected scene generator was extended to provided the expected strength of edges and expected greyscale of faces, then PSEIKI's scheduler could be extended to use these values in its KS scheduling algorithms. By spending most of its processing resources on matching the easily recognizable elements, (e.g. strong edges and faces with a high contrast to their surrounding faces) PSEIKI should be able to narrow the focus of its search much more efficiently.

PSEIKI could also use more 3D information in its processing. Currently, PSEIKI assumes that all of its data lies on a single plane; the data is either worked on directly in

the image plane or it is backprojected onto the ground ($z = 0$) plane. If the expected scene generator is modified to retain 3D information in the symbolic descriptions of the expected scene, then it may be possible to backproject the data into the world coordinate frame in a more intelligent fashion. For example, the system could be enlarged by adding a new *backprojection* KS that would use the 3D geometric information stored in the expected scene to build a 3D model of the observed image. This 3D model then could be considered without the distortions due to prospective projection. One technique that this KS could use to build the model is described in [MulSha85].

PSEIKI also could be extended to handle range data. This extension also would rely on the availability of 3D expected scene information. PSEIKI was designed with this extension in mind; therefore, only the metrics described in chapter 6 need to be changed to allow PSEIKI to work with range data. For example, the face-level compatibility metrics could be extended to handle 3D data by using the directions of the normal vectors of the two faces. In the three dimensional case, these metrics could be defined as

$$\text{colocate}_{3D}(F_1, F_2) = \frac{D_{max} - D_{centroid}}{D_{max}} \times \cos(\theta)$$

and

$$\text{noncolocate}_{3D}(F_1, F_2) = \frac{D_{centroid}}{D_{max}} \times \sin(\theta)$$

where the distance parameters are defined as before and θ is the acute angle between the two normal vectors. The definitions of the relational constraint transformations also would need to be extended to include a rotational component that would make the faces' normal vectors collinear.

Another possibility would be to extend PSEIKI to perform model-based sensor fusion. That is, it might be possible to merge the information from a number of sensors by using cues taken from the expected scene. Then, a new implementation of PSEIKI would include a variable number of data panels each storing the data from a single sensor. Processing similar to that being performed by the current version of PSEIKI could be performed to match data from each data panel with the model data. The complementary nature of the data derived from the various sources might make the blackboard processing more robust. For example, it might be possible to merge information from image data and range data of the same scene by using a three panel version of PSEIKI -- one panel for model data, one panel for image data and one panel for range data. It also

might be possible to merge, into a single interpretation of the observed scene, the edges found by the edge-based preprocessor with the regions found by the region-based preprocessor.

Finally, one of the most theoretically interesting areas for further investigation concerns the propagation of belief values up the hierarchy. We do not know how our choice of the consistency metric for an element's children (the updating SEF from the most believed child) affects the element's belief function or those of its ancestors. An in-depth investigation of this propagation scheme, or the development of a competing scheme would certainly be a major contribution to the field.

BIBLIOGRAPHY

BIBLIOGRAPHY

[AhoHop74] Aho, A. V., J. E. Hopcraft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms,* Addison-Wesley, Reading, Mass., 1974.

[AndKak87a] Andress, K. M. and A. C. Kak, "A Production System Environment for Integrating Knowledge with Vision Data," *Proc. of the 1987 AAAI Workshop on Spatial Reasoning and Multi-Sensor Integration*, pp. 1-12, Morgan-Kaufmann Publishers, Inc., 1987.

[AndKak87b] Andress, K. M. and A. C. Kak, "PSEIKI: A Production System Environment for Integrating Knowledge with Images," School of Electrical Engineering, Purdue University, Technical Report TR-EE 87-36, 1987.

[AndKak88a] Andress, K. M. and A. C. Kak, "Evidence Accumulation and Flow of Control in a Hierarchical Spatial Reasoning System," *The AI Magazine*, vol. 9, no. 2, pp. 75-95, 1988.

[AndKak88b] Andress, K. M. and A. C. Kak, "The PSEIKI Report -- Version 2," School of Electrical Engineering, Purdue University, Technical Report TR-EE 88-9, 1988.

[BakBin81] Baker, H. H. and T. O. Binford, "Depth from Edge and Intensity based Stereo," *Proceedings IJCAI*, vol. 6, pp. 631-636, 1981.

[BalBro82] Ballard, D. H. and C. M. Brown, *Computer Vision,* Prentice Hall, Englewood Cliffs, New Jersey, 1982.

[BarTho81] Barnard, S. T. and W. B. Thompson, "Disparity Analysis of Images," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-2, pp. 333-340, 1981.

[Bar83] Barnard, S. T., "Interpreting Perspective Images," *Artificial Intelligence*, vol. 21, pp. 435-462, 1983.

[Bar81] Barnett, J. A., "Computational Methods for a Mathematical Theory of Evidence," *Proceedings IJCAI*, pp. 868-875, 1981.

[BarTen81] Barrow, H. G. and J. M. Tenenbaum, "Interpreting Line Drawings as Three-Dimensional Surfaces," *Artificial Intelligence*, vol. 17, pp. 75-116, 1981.

[BesJai85] Besl, P. J. and R. C. Jain, "Three-Dimensional Object Recognition," *ACM Computing Surveys*, vol. 17, no. 1, pp. 75-144, 1985.

[Bes88] Besl, P. J., "Geometric Modeling and Computer Vision," *Proceedings of the IEEE*, vol. 76, no. 8, pp. 936-958, 1988.

[Bin82] Binford, T. O., "Survey of Model-Based Image Analysis Systems," *The International Journal of Robotics Research*, vol. 1, no. 1, pp. 18-64, MIT Press, 1982.

[BinLev87] Binford, T. O., T. S. Levitt, and W. B. Mann, "Bayesian Inference in Model-Based Machine Vision," *Proceedings AAAI Workshop on Uncertainty in Artificial Intelligence*, Seattle, July 1987.

[Bla89] Blask, S. G., "IHSE: An Interactive Hierarchical Scene Editor," RVL Memo #11, Robot Vision Lab, EE Building, Purdue University, 1989.

[BriFen70] Brice, C. R. and C. L. Fennema, "Scene Analysis Using Regions," *Artificial Intelligence*, vol. 1, no. 3, pp. 205-226, 1970.

[Bro81] Brooks, R. A., "Symbolic Reasoning Among 3-D Models and 2-D Images," *Artificial Intelligence*, vol. 17, pp. 285-348, 1981.

[BroFar85] Brownston, L., R. Farrell, E. Kant, and N. Martin, *Programming Expert Systems in OPS5*, Addison-Wesley, 1985.

[ConMun87] Connolly, C. I., J. L. Mundy, J. R. Stenstrom, and D. W. Thompson, "Matching from 3-D Range Models into 2-D intensity Scenes," *Proceedings on 1st International Conference on Computer Vision*, pp. 65-72, 1987.

[CorGal87] Corkill, D. D., K. Q. Gallagher, and P. M. Johnson, "Achieving Flexibility, Efficiency, and Generality in Blackboard Architectures," *Proceedings AAAI*, 1987.

[CroSan87] Crowley, J. L. and A. C. Sanderson, "Multiple Resolution Representation and Probabilistic Matching of 2-D Grey-Scale Shape," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-9, no. 1, pp. 113-121, 1987.

[CroRam87] Crowley, J. L. and F. Ramparany, "Mathematical Tools for Representing Uncertainty in Perception," *Proc. of the 1987 AAAI Workshop on Spatial Reasoning and Multi-Sensor Integration*, pp. 293-302, Morgan-Kaufmann Publishers, Inc., 1987.

[DubPra85] Dubois, D. and H. Prade, "Combination and Propagation of Uncertainty with Belief Functions," *Proceedings IJCAI*, 1985.

[DubPra86] Dubois, D. and H. Prade, "Set-theoretic Operations on Bodies of Disjunctive or Conjunctive Evidence," *Proceedings of the North American Fuzzy Information Processing Society*, pp. 107-124, New Orleans, 1986.

[DudHar73] Duda, R. O. and P. E. Hart, *Pattern Classification and Scene Analysis*, Wiley, New York, 1973.

[DugBob85] Dugan, J. B., A. Bobbio, G. Ciardo, and K. Trivedi, "The Design of a Unified Package for the Solution of Stochastic Petri Net Models," *Proceedings of International Conference on Timed Petri Nets*, 1985.

[DurLes87] Durfee, E. H., V. R. Lesser, and D. D. Corkill, "Coherent Cooperation Amoung Communicating Problem Solvers," *IEEE Transactions on Computers*, vol. Vol C-36, no. 11, pp. 1275-1291, 1987.

[Ebe76] Eberlein, R. B., "An iterative Gradient Edge Detection Algorithm," *Computer Graphics and Image Processing*, vol. 5, pp. 245-253, Acedemic Press, 1976.

[EngMor88] Engelmore, R. and T. Morgan, *Blackboard Systems*, Addison Wesley, Wokingham, England, 1988.

[ErmHay80] Erman, L. D., F. Hayes-Roth, V. R. Lesser, and D. R. Reddy, "The Hearsay-II Speech Understanding System: Integrating Knowledge to Resolve Uncertainty," *ACM Computing Surveys*, pp. 213-253, 1980.

[ErmLon81] Erman, L. D., P. E. London, and S. F. Fickas, "The Design and an Example Use of Hearsay-III," *Proceedings Joint Int'l Conference on Artificial Intelligence*, 1981.

[FauPra79] Faux, I. D. and M. J. Pratt, *Computational Geometry for Design and Manufacturing*, Ellis Horwood, Ltd., New York, 1979.

[FenLes77] Fennell, R. D. and V. R. Lesser, "Parallelism in Artificial Intelligence Problem Solving: A Case Study of Hearsay II," *IEEE Transactions on Computers*, vol. Vol C-26, no. 2, pp. 98-111, 1977.

[FolVan82] Foley, J. D. and A. VanDam, *Fundamentals of Interactive Computer Graphics*, Addison Wesley, Reading, Mass., 1982.

[For86] Forgy, C. L., *OPS/83 User's Manual and Report*, Production Systems Technologies, Inc., 1986.

[GarJoh79] Garey, M. R. and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman and Company, New York, 1979.

[GarCor87] Garvey, A., C. Cornelius, and B. Hayes-Roth, "Computational Costs versus Benefits of Control Reasoning," *Proceedings AAAI*, 1987.

[GarLow81] Garvey, T. D., J. D. Lowrance, and M. A. Fischler, "An Inference Technique for Integrating Knowledge from Disparate Sources," *Proceedings IJCAI*, pp. 319-325, 1981.

[GorSho85] Gordon, J. and E. H. Shortliffe, "A Method for Managing Evidential Reasoning in a Hierarchical Hypothesis Space," *Artificial Intelligence*, vol. 26, pp. 323-357, 1985.

[Gri81a] Grimson, W. E. L., "A computer implementation of a theory of human stereo vision," *Phil. Trans. Roy. Soc. Lon.*, vol. B 292, pp. 217-253, 1981.

[Gri81b] Grimson, W. E. L., *From Images to Surfaces: A Computational Study of the Human Early Visual System*, MIT Press, 1981.

[HanRis78] Hanson, A. R. and E. M. Riseman, "VISIONS: A Computer System for Interpreting Scenes," in *Computer Vision Systems*, ed. E. M. Riseman, pp. 303-333, Academic Press, 1978.

[HarSha85] Haralick, R. M. and L. G. Shapiro, "SURVEY: Image Segmentation Techniques," *Computer Vision, Graphics, and Image Processing*, vol. 29, pp. 100-132, Academic Press, 1985.

[HarMar85] Hartquist, E. E. and H. A. Marisa, "PADL-2 Users Manual," Production Automation Project, The University of Rochester, UM-10/2.1, 1985.

[Hay85] Hayes-Roth, B., "A Blackboard Architecture for Control," *Artificial Intelligence*, pp. 251-321, 1985.

[HayLes77] Hayes-Roth, F. and V. R. Lesser, "Focus of Attention in the Hearsay-II Speech Understanding System," *Proceedings IJCAI*, 1977.

[HofJai87] Hoffman, R. and A. K. Jain, "Segmentation and Classification of Range Images," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-9, no. 5, pp. 608-620, 1987.

[HorPav76] Horowitz, S. L. and T. Pavlidis, "Picture Segmentation by a Tree Traversal Algorithm," *Journal of the ACM*, vol. 23, no. 2, pp. 368-388, April 1976.

[HuSto87] Hu, Gongzhu and George Stockman, "3-D Scene Analysis Via Fusion of Light Striped Image and Intensity Image," *Proc. of the 1987 AAAI Workshop on Spatial Reasoning and Multi-Sensor Integration*, pp. 138-147, Morgan-Kaufmann Publishers, Inc., 1987.

[HunJay87] Huntsberger, T. L. and S. N. Jayaramamurthy, "A Framework for Multi-Sensor Fusion in the Presence of Uncertainty," *Proceedings of the 1987 AAAI Workshop on Spatial Reasoning and Multi-Sensor Fusion*, Morgan-Kaufmann Publishers, Inc., 1987.

[HwaDav85] Hwang, V. S. S., L. S. Davis, and T. Matsuyama, "Hypothesis Integration in Image Understanding Systems," *Computer Vision, Graphics, and Image Processing*, vol. 36, pp. 321-371, Academic Press, 1985.

[JohHay87] Johnson, M. V. Jr. and B. Hayes-Roth, "Intergrating Diverse Reasoning Methods in the BB1 Blackboard Control Architecture," *Proceedings AAAI*, 1987.

[KakRob87] Kak, A. C., B. A. Roberts, K. M. Andress, and R. L. Cromwell, "Experiments in the Integration of World Knowledge with Sensory Information for Mobile Robots," *Proceedings IEEE International Conference on Robotics and Automation*, vol. 2, pp. 734-741, 1987.

[Kim88] Kim, W. Y. and A. C. Kak, "A Cross Scanning Structured Light System for 3-D Robot Vision," School of Electrical Engineering, Purdue University, Technical Report (in preparation), 1988.

[Koh47] Kohler, W., *Gestalt Pshychology*, Liverright, Ney York, 1947.

[KreMik89] Kretsch, J. L. and E. M. Mikhail, "Applications of Artificial Intelligence to Digital Photogrammetry," School of Civil Engineering, Purdue University, Technical Report CE-PH 89-1, 1989.

[Kui77] Kuipers, B., "Representing Knowledge of Large-Scale Space," MIT AI Lab Technical Report AI-TR-418, July 1977.

[Kyb87] Kyburg, H. E. Jr., "Bayesian and Non-Bayesian Evidential Updating," *Artificial Intelligence*, pp. 271-293, 1987.

[LawMcC87] Lawton, D. T. and C. C. McConnell, "Perceptual Organization Using Interestingness," *Proc. of the 1987 AAAI Workshop on Spatial Reasoning and Multi-Sensor Integration*, pp. 405-419, Morgan-Kaufmann Publishers, Inc., 1987.

[LehRey86] Lehrer, N. B., G. Reynolds, and J. Griffith, "A Method for Initial Hypothesis Formation in Image Understanding," *Proc. of the 1st IEEE Conference on Computer Vision*, 1986.

[LesErm77] Lesser, V. R. and L. D. Erman, "A retrospective View of the Hearsay-II Architecture," *Proceedings Joint Int'l Conference on Artificial Intelligence*, 1977.

[LesErm80] Lesser, V. R. and L. D. Erman, "Distributed Interpretation: A Model and Experiment," *IEEE Transactions on Computers*, vol. Vol C-29, no. 12, pp. 1144-1289, 1980.

[LevLaw87] Levitt, T. S., D. T. Lawton, D. M. Chelberg, and P. C. Nelson, "Qualitative Navigation," *Proc. of DARPA Image Understanding Workshop*, Los Angeles, Febuary 1987.

[LopKak89] Lopez-Abadia, C. and A. C. Kak, "Vision Guided Mobile-Robot Navigation," School of Electrical Engineering, Purdue University, Technical Report TR-EE 89-34, 1989.

[Law85] Lowe, D. G., *Perceptual Organization and Visual Recognition*, Kluwer Academic , Boston, 1985.

[MarPog79] Marr, D. and T Poggio, "A Theory of Human Stereo Vision," *Proc. R. Soc. Lond.*, vol. B, no. 204, pp. 301-328, 1979.

[Mar82] Marr, D., *Vision*, W. H. Freeman and Co., 1982.

[MarCon84] Marsan, M. A., G. Conte, and G. Balbo, "A Class of Generalized Stochastic Petri Nets for the Performance Evaluation of Multiprocessor Systems," *ACM Transactions on Computer Systems*, vol. 2, no. 2, 1984.

[Mas87] Mashburn, T., "A Polygonal Solid Modeling Package," M.S. Thesis, School of Mechanical Engineering, Purdue University, 1987.

[MatHwa85] Matsuyama, T. and V. S. S. Hwang, "SIGMA: a Framework for Image Understanding - Integration of Bottom-up and Top-down Analysis," *Proceedings IJCAI*, pp. 908-915, 1985.

[MckHar85] McKeown, D. M. Jr., W. A. Harvey, Jr., and J. McDermott, "Rule-Based Interpretation of Aerial Imagery," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-7, no. 5, pp. 570-585, 1985.

[MedNev84] Medioni, G. and R. Nevatia, "Matching Images Using Linear Features," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-6, pp. 675-685, 1984.

[MitHar87] Mitchell, D. H., S. A. Harp, and D. K. Simkin, "A Knowledge Engineer's Comparison of Three Evidence Aggregation Methods," *Proceedings of Uncertainty in Artificial Intelligence Workshop*, pp. 297-312, 1987.

[Mol82] Molloy, M. K., "Performance Analysis Using Stochatic Petri Nets," *IEEE Transactions on Computers*, vol. C-31, no. 9, 1982.

[Mor85] Mortenson, M. E., *Geometric Modeling*, John Wiley & Sons, New York, 1985.

[Mor66] Morton, G. M., "A computer oriented geodetic data base and a new technique in file sequencing," unpublished, IBM, Ottawa, Canada, 1966.

[MulSha85] Mulganonkar, P. G. and L. G. Shapiro, "Hypothesis-Based Geometric Reasoning About Persepective Images," *Proceedings of the Third Workshop on Computer Vision: Representation and Control*, IEEE Computer Society Press, 1985.

[NagMat80] Nagao, M. and T. Matsuyama, *A Structural Analysis of Complex Aerial Photographs*, Plenum Press, 1980.

[NazLev84] Nazif, A. M. and M. D. Levine, "Low Level Segmentation: An Expert System," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-6, no. 5, pp. 555-577, 1984.

[NavBab80] Nevatia, R. and K. R. Babu, "Linear Feature Extraction and Description," *Computer Graphics and Image Processing*, vol. 13, pp. 257-269, 1980.

[NiiFei78] Nii, H. P. and E. A. Feigenbaum, "Rule-based Understanding of Signals," in *Pattern Directed Inference Systems*, ed. F. Hayes-Roth, pp. 53-68, Academic Press, 1978.

[Nii86b] Nii, P. H., "Blackboard Systems from a Knowledge Engineering Perspective," *The AI Magazine*, pp. 82-106, August 1986.

[Nii86a] Nii, P. H., "Blackboard Systems: The Blackboard Model of Problem Solving and the Evolution of the Blackboard Architectures," *The AI Magazine*, pp. 38-53, Summer 1986.

[Nil80] Nilsson, N. J., *Principles of Arificial Intelligence*, Tioga Publishing Company, Palo Alto, CA, 1980.

[Pea84] Pearl, J., *Heuristics*, Addison Wesley, Reading, Mass., 1984.

[Pea86] Pearl, J., "Fusion, Propagation, and Structuring in Bayesian Networks," *Artificial Intelligence*, 1986.

[Pet81] Peterson, J. L., *Petri Net Theory and the Modeling of Systems*, Prentice-Hall, Englewood Cliffs, NJ, 1981.

[RamHo80] Ramamoorthy, C. V. and G. S. Ho, "Performance Evaluation of Asynchronous Concurrent Systems Using Petri Nets," *IEEE Transactions on Software Engineering*, vol. SE-6, no. 5, 1980.

[ReyStr86] Reynolds, G., D. Strahman, N. Lehrer, and L. Kitchen, "Plausible Reasoning and the Theory of Evidence," COINS Tech. Report 86-11, University of Massachussetts, 1986.

[RosKak82] Rosenfeld, A. and A. C. Kak, *Digital Picture Processing, Vols. 1 & 2*, Academic Press, Orlando, 1982.

[SafGot90] Safranek, R. J., S. Gottschlich, and A. C. Kak, "Evidence Accumulation Using Binary Frames of Discernment for Verification Vision," *To appear in IEEE Trans. Robotics and Automation*, 1990.

[Sam84b] Samet, H., "A Tutorial on Quadtree Research," in *Multiresolution Image Processing and Analysis*, Springer-Verlag, 1984.

[Sam84a] Samet, H., "The Quadtree and Related Hierarchical Data Structures," *ACM Computing Surveys*, vol. 16, no. 2, ACM, June 1984.

[Sed84] Sedgewick, R., *Algorithms*, Addison Wesley, Reading, Mass., 1984.

[Sha76] Shafer, G., *A Mathematical Theory of Evidence*, Princeton University Press, 1976.

[ShaLog87] Shafer, G. and R. Logan, "Implementing Dempster's Rule for Hierarchical Evidence," *Artificial Intelligence*, vol. 33, pp. 271-298, 1987.

[ShaHar81] Shapiro, L. G. and R. M. Haralick, "Structural Descriptions and Inexact Matching," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-3, no. 5, pp. 504-519, 1981.

[Sme76] Smets, P., "Combining Non-Distinct Evidences," *Proceedings North American Fuzzy Information Processing Society*, pp. 544-548, New Orleans, 1986.

[Tri82] Trivedi, K. S., *Probability & Statistics with Reliability, Queuing, and Computer Science Applications*, Prentice-Hall, Englewood Cliffs, NJ, 1982.

[VoeReq77] Voelcker, H. B. and A. A. G. Requicha, "Geometric Modeling of Mechanical Parts and Processes," *IEEE Computer*, 1977.

[Wat70] Watkins, G. S., "A Real-Time Visible Surface Algorithm," Univeristy of Utah Computer Science Department, UTEC-CSc-70-101, 1970.

[WatArv84] Watson, L. T., K. Arvind, R. W. Ehrich, and R. M. Haralick, "Extraction of Lines and Regions From Grey Tone Line Drawings," *Pattern Recognition*, vol. 17, no. 5, pp. 493-507, 1984.

[WatArv87] Watson, L. T., K. Arvind, R. W. Ehrich, and R. M. Haralick, "Extraction of Lines and Regions From Grey Tone Line Drawing Images," *Pattern Recognition*, vol. 17, no. 5, 1987.

[WitTen83a] Witkin, A. P. and J. M. Tenenbaum, "On the Role of Structure in Vision," in *Human and Machine Vision*, ed. Rosenfeld, pp. 481-543, Academic Press, New York, 1983.

[WitTen83b] Witkin, A. P. and J. M. Tenenbaum, "What is Perceptual Organization For?," *Proceedings IJCAI*, pp. 1023-1026, 1983.

[YanKak86] Yang, H. S. and A. C. Kak, "Determination of the Identity, Position and and Orientation of the Topmost Object in a Pile.," *Computer Vision, Graphics, and Image Processing*, vol. 36, pp. 229-255, Academic Press, 1986.

[Yen86] Yen, J., "A Reasoning Model Based on an Extended Dempster-Shafer Theory," *Proceedings AAAI*, pp. 125-131, 1986.

[Zis78] Zisman, M. D., "Use of Production Systems for Modeling Asynchronous, Concurrent Processes," in *Pattern Directed Inference Systems*, ed. F. Hayes-Roth, pp. 53-68, Academic Press, 1978.

[Zub85] Zuberek, W. M., "Performance Evaluation Using Extended Timed Petri Nets," *Proceedings of International Conference on Timed Petri Nets*, 1985.

[Zuc76] Zucker, S. W., "Region Growing: Childhood and Adolescence," *Computer Graphics and Image Processing*, vol. 5, pp. 382-399, 1976.

APPENDICES

APPENDIX A

A MONTE CARLO INVESTIGATION OF THE ROBUSTNESS AND EFFICIENCY OF THE LABEL-BASED ACCUMULATION PROCEDURE

As described in chapter 5, the label-based accumulation scheme achieves its efficiency by restricting the amount of evidence accumulated into a belief function. Therefore, in systems using an evidence accumulation scheme based on model 1, one cannot hope that the new scheme would perform as well as one which accumulated all available evidence into the belief function. However, it may be advantageous to use the label-based accumulation procedure for efficiency reasons. Furthermore, if the new scheme performs satisfactorily in an application, then any marginal increase in performance achieved by accumulating the remaining evidence into the belief function may not be needed. It is up to the system designer to decide if the efficiency gained by employing the new accumulation scheme outweighs any decrease in performance.

This appendix describes two Monte Carlo Simulations undertaken to address two main questions. First, how much degradation in the final belief function results from restricting the updating evidence to focus on the label element and its compliment? Second, how much of a computational savings can be achieved by employing the new scheme compared to an accumulation procedure based on the straightforward implementation of Barnett's formulas. Although, strictly speaking, the results of these simulations apply to a system only if the system's belief functions have the same statistics as those used in the simulation, these simulations show that the label-based accumulation procedure is a viable alternative to the baseline accumulation procedure for some applications. The simulations presented in this appendix are not appropriate for systems using evidence accumulation schemes based on model 2, because the simulations are based on the

unrealistic assumption that it is possible to accumulate all available evidence into the belief function. Because PSEIKI uses the second model for label-based evidence accumulation, these simulations do not pertain strictly to PSEIKI. However, the simulations are presented in this document for completeness' sake.

Simulation 1:

The first investigation, using Monte Carlo simulation techniques, was carried out to compare the accuracy of the label-based accumulation scheme with the accuracy of the baseline accumulation scheme (one in which all available information is accumulated into the belief function). This simulation was undertaken to determine the performance of the label-based accumulation scheme as the statistics of the input belief functions were varied. Each of the composite belief functions had two elements in its FOD, $\Theta = \{1, 2\}$. Without loss of generality, the first element in the FOD was defined to be the correct label element. In each trial, the data from 10 composite belief functions were combined into a final belief function using the baseline and the label-based schemes. At the end of each trial, if the label was assigned to the first element in the FOD, then the label was said to be correct; otherwise, it was said to be incorrect. The probability masses contained in each composite belief function was generated with random data. For each experiment in the simulation, the statistics of the random data were held fixed (the same statistics were used to generate the random data for all 10 belief functions). The statistics were varied across the experiments to determine what effect the distributions had on the results. Eighty one experiments were run in this simulation (one for each point in the graphs shown in Figs. A.2 - A.4). To assure statistically valid results, each experiment consisted of 10,000 trials.

To limit the complexity of the investigation and to aid in the visualization of the results, only the mean values for the evidence confirming both members of the FOD, $E\{M^1(1)\}$ and $E\{M^2(2)\}$, were explicitly set at the start of each experiment (where $E\{\cdot\}$ denotes the expected value of its argument). These two parameters ranged from 0.1 to 0.9 in increments of 0.1. The parameters varied across experiments and were fixed for any given experiment. Because each of the two parameters could assume one of 9 possible values, the first simulation contained a total of 81 experiments. All other parameters for the random functions used to generate the input belief functions were calculated based on the two user specified parameters. For example, the mean values of the confirmatory and disconfirmatory probability masses for each SEF were defined to have

unity sum.† This is denoted formally as:

$$E\{M^i\}(\neg i) = 1.0 - E\{M^i(i)\} \tag{A.1}$$

The variance of the underlying normal distributions was a function of their mean values. The following equation shows the function used to determine the variance of a mass function given its mean value.

$$\sigma^2 = \begin{cases} \frac{1}{2}(E\{M^i(\cdot)\}) & \text{if } E\{M^i(\cdot)\} \le 0.5 \\ \frac{1}{2}(1.0 - E\{M^i(\cdot)\}) & \text{if } E\{M^i(\cdot)\} > 0.5 \end{cases} \tag{A.2}$$

where σ^2 is the variance of the confidence function. Fig. A.1 shows how the variance of the Gaussian density functions changed as a function of the mean value. Notice that the variance becomes smaller as the mean value approaches either zero or one. Setting the variance in this manner guarantees that most of the values produced by the random number generator will fall between zero and one, regardless of the mean of the density function. Table A.1 shows an example of the mass values for a single trial of this simulation. It shows all the masses for all 10 composite belief functions.

The mass initialization process also bounds the masses assigned to the SEFS to between 0.05 and 0.95. The masses are bounded to prevent the belief function from *saturating* as described in [SafGot90]. A belief function is said to saturate when the mass for a singleton proposition approaches one; the belief function is said to be completely saturated when the mass is exactly equal to one (i.e. the proposition holds the entire mass for the belief function). Because of the renormalization associated with Dempster's rule, the combination of a saturated belief function with another belief function will always produce a saturated belief function with the same focal element, unless the two input belief functions are completely contradictory. Theoretically, a saturated belief function cannot be produced by the combination of two nonsaturated belief functions; however, this may occur in computer-based implementations because of the finite precision associated with floating-point numbers. To prevent the masses in the overall belief function from saturating, any probability mass from the 10 input belief functions

† The fact that the mean values for the confirmatory and disconfirmatory evidence have unity sum does not imply that the individual SEFs will have no uncertainty. The random nature of the mass initialization process guarantees that the SEFs will have some uncertainty (because the random number generator generated values below the mean value).

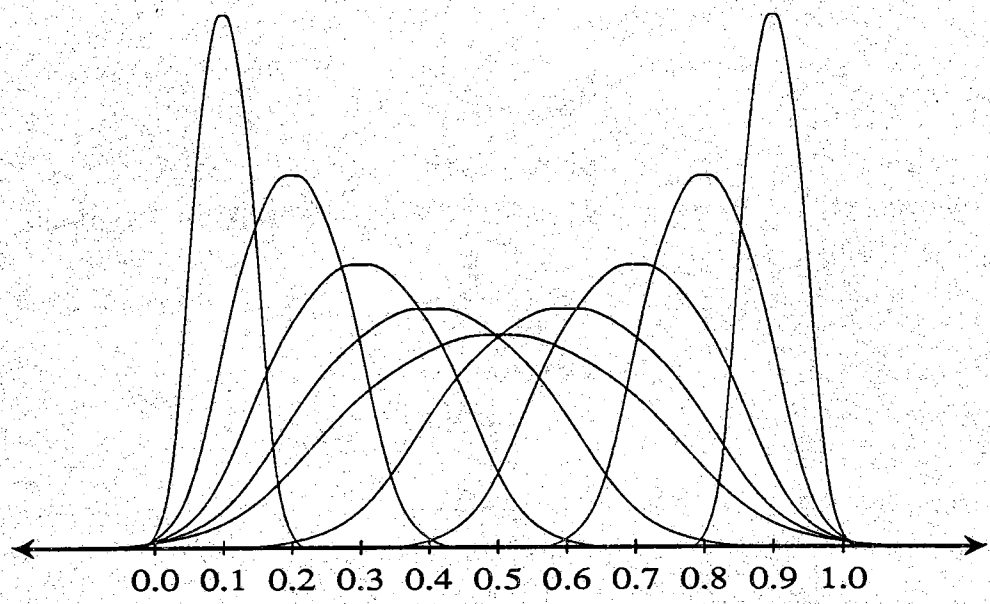


Figure A.1 This figure shows how the variance of the Gaussian density functions changed as a function of the mean value.

not between 0.05 and 0.95 was discarded and reinitialized.

Finally, the SEFs were checked to guarantee that they were valid belief functions. If the sum of the confirmatory and disconfirmatory evidence in a SEF summed to less than one, then the remaining mass was assigned to the FOD to represent the SEF's uncertainty. However, if the sum of the mass was greater than one, then both the confirmatory and disconfirmatory masses were discarded and new masses were generated.

At the end of each trial, relevant parameters were determined and accumulated into the total statistical pool. These parameters include the percentage of trials in which the final label was correct (was equal to the first element in the FOD), the average belief and disbelief in the final label and the average number of SEF combinations used to determine the final label. Fig. A.2 shows the percentage of trials in which the final label was correct for the baseline and label-based accumulation schemes. Because the overall goal of the system is defined to be the determination of the correct label, the percentage of correct labelings is a good measure of the accuracy of the accumulation scheme. The left panel shows the percentage of trials in which the final label was correct when the baseline accumulation scheme was used. The right panel shows the percentage of trials in

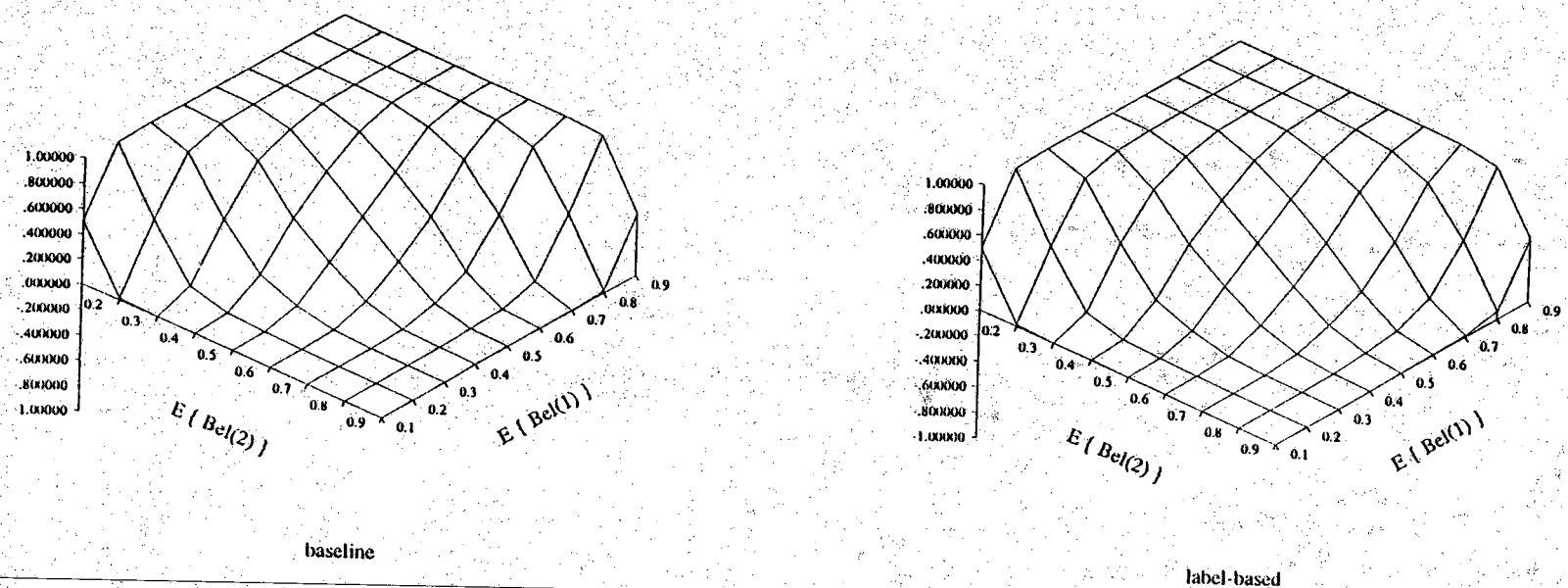


Figure A.2 This figure shows a plot of the percentage of trials in which the final belief function's label was correct as E{Bel(1)} and E{Bel(2)} were varied. Each of the 10 belief functions accumulated had 2 elements in their FOD. The left panel shows the percentage of trials in which the final label was correct when the baseline accumulation scheme was used. The right panel shows the percentage of trials in which the final label was correct when the label-based accumulation scheme was used. The panel on the following page shows the difference between the two graphs.

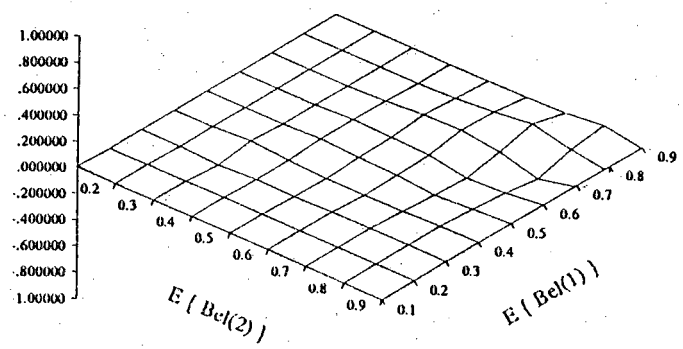


Figure A.2, continued.

Table A.1 This table shows an example of the mass values for a single trial of the first simulation. It shows all the masses for all 10 composite belief functions. Note that each row corresponds to a composite belief function as defined in chapter 5. To remind the reader, for the i^{th} belief function, $M_i^1(1)$, $M_i^1(\neg 1)$ and $M_i^1(\Theta) = 1.0 - M_i^1(1) - M_i^1(\neg 1)$ constitutes a simple evidence function focused on element 1. One Monte Carlo experiment consisted of generating 10,000 sets of 10 belief functions like those shown here, combining each set by either the baseline accumulation procedure or our label-based accumulation procedure and then determining the percentage of times that the final label is correct.

$E\{M^1(1)\} = 0.1$ $E\{M^2(2)\} = 0.3$				
i	$M_i^1(1)$	$M_i^1(\neg 1)$	$M_i^2(2)$	$M_i^2(\neg 2)$
1	0.087487	0.901283	0.114168	0.717069
2	0.066644	0.931318	0.358946	0.531399
3	0.109026	0.869632	0.223079	0.590347
4	0.055742	0.906334	0.277684	0.449074
5	0.082370	0.894598	0.412074	0.569845
6	0.101785	0.853475	0.171803	0.639343
7	0.082065	0.905981	0.192292	0.408769
8	0.154390	0.844661	0.245509	0.497720
9	0.062120	0.892161	0.329408	0.474937
10	0.082053	0.857212	0.219839	0.666866

which the label-based procedure produced the correct label. The last panel shows the difference between the two graphs (baseline minus label-based). As can be seen in these plots, the label-based accumulation scheme correctly assigned the final label nearly as often as the baseline accumulation scheme. On the average, the baseline scheme assigned 0.02% more final belief functions with the correct labeled than the label-based scheme.

The belief attached to the final label also is a good metric of the accumulation scheme's accuracy, although this measure is not as important as the percentage of correct labelings. If a label is correct, one would hope for maximal belief; conversely, if the final label is incorrect, one would hope for minimal belief. Figs. A.3 and A.4 show the average belief in the final label for correct and incorrect labels, respectively. In both figures, the left panel shows the average belief in a label when the baseline accumulation scheme was used to combine the belief functions. The right panel shows the average belief in a label when the label-based accumulation scheme was used. The last panel shows the difference in the average belief between the two accumulation schemes (label-based minus baseline). As can be seen in these figures, the label-based scheme lends approximately the same belief to the final label as the baseline scheme does. However, the label-based scheme does have a slight tendency to lend greater belief to both correct and incorrect labels. Because the belief in the label is of secondary concern (the correctness of the label is the primary concern), the slight difference in the belief attached to the final label by the two schemes does not seem to be a problem.

Simulation 2:

The second Monte Carlo simulation was undertaken to compare the computational efficiency of the baseline and the label-based accumulation schemes as the problem size varied. In this simulation, the statistics of the random number generator were held fixed for all experiments and the problem size was varied. The two measures of input problem size used in this simulation were the number of belief functions combined into the final belief function and the number of elements in their FODs. The measure of computational complexity used in this simulation was the number of SEF combinations needed by an accumulation procedure to arrive at the final belief function.

In this simulation, the number of input belief functions and FOD size were varied from 5 to 50 in increments of 5. These two parameters varied across experiments and were fixed for any given experiment. Because each of the two parameters could assume

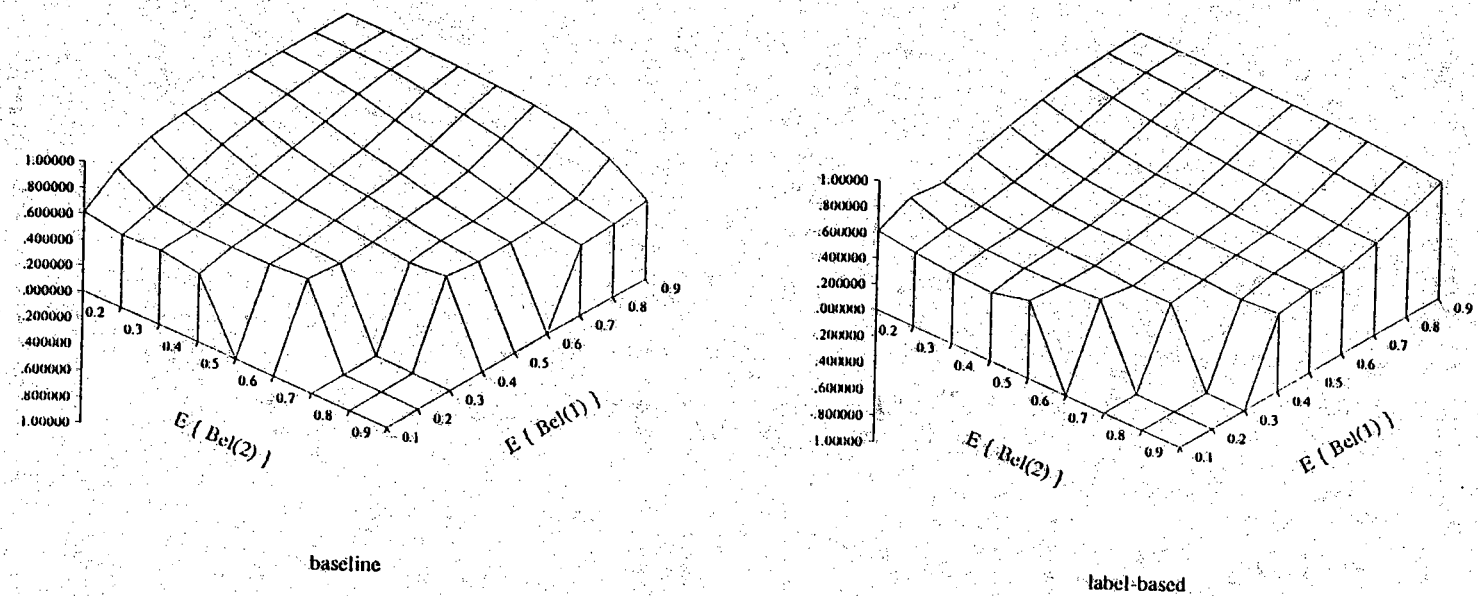


Figure A.3 This figure shows plots of the average belief in a final belief function's label if the label was correctly assigned. (i.e. the label was assigned to the first element in the FOD). In this experiment, 10 belief functions with 2 elements in each of their FODs were combined and E{Bel(1)} and E{Bel(2)} were varied. The left panel shows the average belief when the baseline accumulation scheme was used. The right panel shows the average belief when the label-based accumulation scheme was used. The panel on the next page shows the difference between the two graphs.

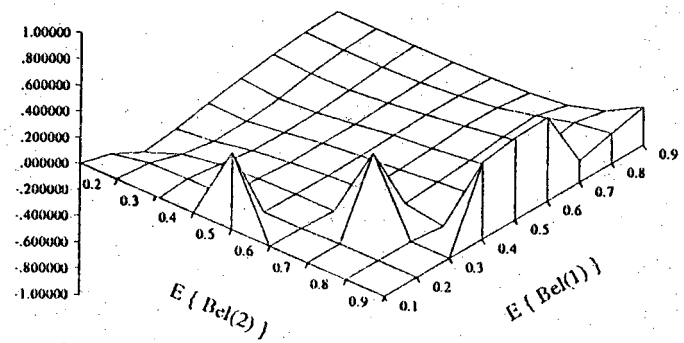


Figure A.3, continued.

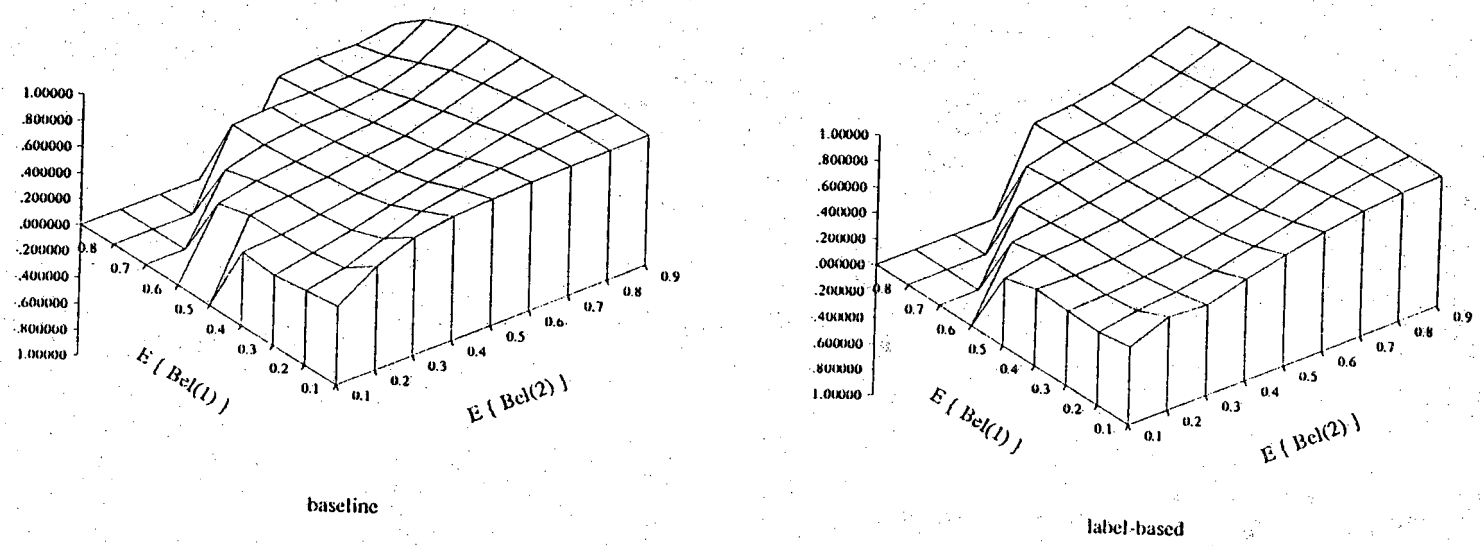


Figure A.4 This figure shows plots of the average belief in a final belief function's label if the label was incorrectly assigned. (i.e. the label was not assigned to the first element in the FOD). In this experiment, 10 belief functions with 2 elements in each of their FODs were combined and E{Bel(1)} and E{Bel(2)} were varied. The left panel shows the average belief when the baseline accumulation scheme was used. The right panel shows the average belief when the label-based accumulation scheme was used. The panel on the next page shows the difference between the two graphs.

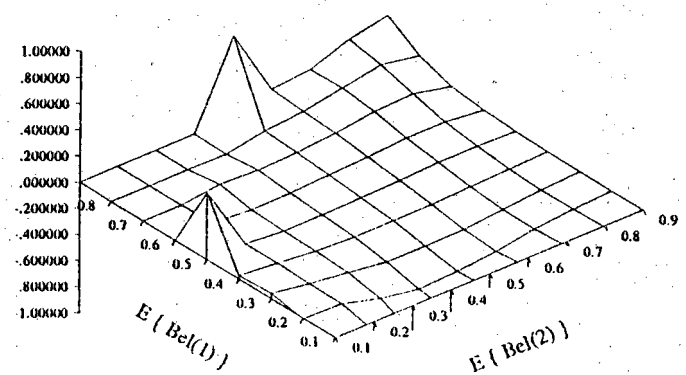


Figure A.4, continued.

one of 10 possible values, one hundred experiments were conducted in this simulation (each one corresponding to one point in Fig. A.5).

As in the first simulation, the first element in the FOD was chosen as the correct label element without loss in generality. The probability masses were initialized with random data using the procedure described for the first simulation. That is, the mean values of the confirmatory evidence was specified at the start of every experiment; however, in this simulation, the same mean values were used for all of the experiments. The mean values for the disconfirmatory evidence was determined using equation (A.1) and equation (A.2) was used to set each random function's variance given its mean value. Finally, if any of the random values were invalid (e.g. masses less than zero or greater than one, SEFs with total masses larger than one, etc.), then the invalid values were discarded and new values generated. The confirmatory evidence for each of the SEFs had the following mean values.

$$E\{M^1(1)\} = 0.4$$

$$E\{M^2(2)\} = 0.2$$

$$E\{M^i(i)\} = 0.1 \quad \text{for } i > 2$$

For most problem sizes, the label-based accumulation procedure gave average performance using the above mean values.

Fig A.5 shows the average number of SEF accumulations used to determine the final belief function as the size of the FOD and the number of input belief functions were varied from 5 to 50 in increments of 5. The left panel of Fig. A.5 shows the number of SEF combinations needed by the baseline scheme for varying problem size. The right panel shows the number of SEF combinations needed by the label-based scheme. Again, each point in these graphs represents the average of 10,000 trials. As can be seen from these plots, the computational advantage of the label-based scheme increases rapidly as the size of the problem grows, even though it remains linear with respect to both the number of input belief functions and the size of their FODs.

These two Monte Carlo simulations have demonstrated that it is possible for the label-based accumulation procedure to achieve a large computational gain over the baseline accumulation procedure with only a relatively small loss in accuracy. While these results do depend on the distributions of the input belief functions, it is believed that the new accumulation procedure is a viable alternative in some cases.

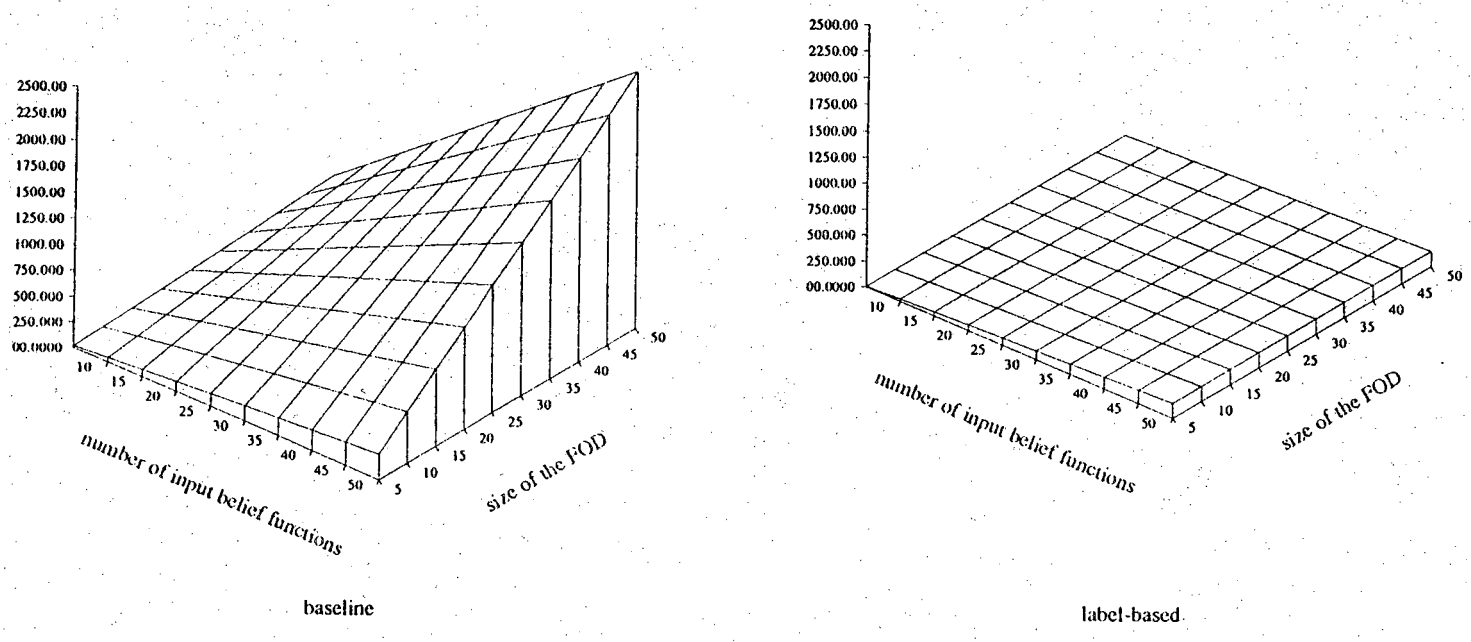


Figure A.5 This figure shows plots of the average number of SEF combinations needed before the accumulation procedure terminated with a final label. In this experiment, The left panel shows the average number of combiations needed when the baseline accumulation scheme was used. The left right shows the average number of combiations needed when the label-based accumulation scheme was used.

APPENDIX B

CONVERSION OF CONFIDENCE VALUES TO BASIC PROBABILITY ASSIGNMENTS

Many systems face the problem of converting raw evidence to a form that is usable by the Dempster-Shafer theory of evidence. Garvey, et. al. were the first to investigate the process of converting raw evidence, such as image feature values, into belief functions [GarLow81]; other work on the conversion of sensor readings to belief functions can be found in [LehRey86], [ReyStr86] and [SafGot90]. In this appendix, a scheme to convert confidence values into a BPA is described. In this scheme a confidence value for any subset of an element's FOD is required to be a value between 0.0 and 1.0. A confidence value of 1.0 for a subset of Θ indicates that the evidence source has conclusive evidence that the element's identity is in that subset. Conversely, a confidence value of 0.0 indicates a lack of evidence that the element's identity is in the subset. To formalize the notion of confidence values, a **confidence function**, Conf, is defined.

$$\text{Conf: } 2^{\Theta} \rightarrow [0, 1]$$

The idea is that the value of this function for any subset represents the amount of evidence provided by a source suggesting that the element's identity is in the subset. Note that this notion is related to the concept of a probability mass in a basic probability assignment; however, a BPA has other properties that are not required of a confidence function. Although a confidence function may not have all the necessary properties of a BPA, a BPA can be defined in terms of an underlying confidence function. To define a BPA, $m(\cdot)$, in terms of a confidence function, it must be defined so that it satisfies three properties.

1) $0.0 \leq m(\cdot) \leq 1.0$

The confidence function meets this criteria by definition.

2) $m(\varnothing) = 0.0$

This property is obtained by setting the probability mass of the null set to zero. This action makes intuitive sense because the null set represents the case in which the element's identity is not a member of the FOD. If this were the case, the FOD would be incomplete and a new, more complete one would be needed.

3) $\sum_{\psi \subseteq \Theta} m(\psi) = 1.0$

This requirement states that the evidence source generating the BPA has unity total-belief. When forming a BPA with this property, the concept of the source's *total confidence* is helpful. A source's total confidence is defined to be

$$Conf_{tot} = \sum_{\substack{\psi \subseteq \Theta \\ \psi \neq \varnothing}} Conf(\psi)$$

This concept can be used to break the problem into three cases.

1) $Conf_{tot} = 1.0$

In this case, the confidence function is a BPA. Therefore, define $m(x) = Conf(x)$ for all $x \in 2^{\Theta}$, $x \neq \varnothing$.

2) $Conf_{tot} < 1.0$

In this case, Conf incompletely specifies the source's belief. A BPA can be defined by assigning the uncommitted portion of the source's belief, its ignorance about the identity of the element, to the entire FOD, Θ.

$$m(x) = \begin{cases} 1.0 - Conf_{tot} & x = \Theta \\ 0.0 & x = \varnothing \\ Conf(x) & \text{else} \end{cases}$$

3) $Conf_{tot} > 1.0$

In this case, the evidence source has over-specified its belief. A BPA is defined by normalizing all the confidence values by its total confidence.

$$m(x) = \frac{Conf(x)}{Conf_{tot}} \quad \text{for all } x \in 2^{\Theta},\ x \neq \varnothing$$

After the preceding operations are applied to the confidence function, a BPA

for the evidence source, $m(\cdot)$, results. Note that defining the BPA in this manner does not affect the validity of the first two requirements for a BPA; this is apparent because $Conf_{tot} \geq Conf(\cdot) \geq 0$.

To see more clearly how the conversion process works, consider the following example. Assume for this example that an evidence source is being used to determine the identity of an object with FOD $\Theta = \{\theta_A, \theta_B, \theta_C, \theta_D\}$. If the evidence source provides non-zero weights only to members of Θ, then the following confidence function might result*:

$$Conf(\theta_A) = 0.7$$
$$Conf(\theta_B) = 0.1$$
$$Conf(\theta_C) = 0.4$$
$$Conf(\theta_D) = 0.05$$

If the total confidence exceeds unity, as in this example, the confidence values are normalized by the summed value resulting in the following BPA over Θ:

$$m(\theta_A) = 0.56$$
$$m(\theta_B) = 0.08$$
$$m(\theta_C) = 0.32$$
$$m(\theta_D) = 0.04$$
$$m(\cdot) = 0.0 \text{ for all other subsets of } \Theta$$

On the other hand, the evidence source could have produced values that sum to less than one, as in the following case:

$$Conf(\theta_A) = 0.7$$
$$Conf(\theta_B) = 0.1$$
$$Conf(\theta_C) = 0.0$$
$$Conf(\theta_D) = 0.05$$

Since the measures now sum to less than unity, there is no reason to normalize. Instead, they are converted directly into a BPA in the following manner:

---

* Note that, in general, an evidence source could provide values to any element of 2^{Θ}, not just elements of Θ.

$m(\theta_A) = 0.7$
$m(\theta_B) = 0.1$
$m(\theta_C) = 0.0$
$m(\theta_D) = 0.05$
$m(\Theta) = 0.15$
$m(\cdot) = 0.0$ for all other subsets of Θ

Note that the amount of belief assigned to Θ is equal to 0.15; this is the difference between unity belief and the evidence source's total confidence. Setting the probability mass in Θ to the difference seems intuitively correct for the simple reason that $Conf(\theta_i)$ is a good measure of the confidence that the object's identity is θ_i. Clearly if the object is not thought to correspond to any of the elements in its FOD to a sufficiently high degree, then some belief may be uncommitted. In the above assignment, $m(\Theta) = 0.15$ represents the uncommitted portion of the belief.

A special case of this conversion process is used in PSEIKI. In PSEIKI, the metrics described in chapter 6 are used as sources of evidence. These metrics provide evidence focusing on singleton propositions and their compliments. The evidence focusing on a particular singleton and its compliment is grouped together to form a confidence function. Thus, when the conversion process is applied to these confidence functions, the resulting belief functions are the simple evidence functions (SEFs) required by Barnett's formulas. To see how the conversion process is used in PSEIKI, consider the following example.

First, assume that a data element, ψ_i, is receiving its initial label. Also assume that the expected-scene metrics have been used to generate evidence focusing on one of its model elements, θ_j, and its compliment. This raw evidence is treated as a confidence function.

$$Conf(\theta_j) = ES\_compatibility(\theta_j, \psi_i)$$

$$Conf(\neg\theta_j) = ES\_incompatibility(\theta_j, \psi_i)$$

If the total confidence of this function is less than 1.0, then the process will produce the following SEF (the masses for all other propositions will be zero).

$$M^{\theta_i}(\theta_i) = \text{ES\_compatibility}(\theta_j, \psi_i)$$

$$M^{\theta_i}(\neg\theta_i) = \text{ES\_incompatibility}(\theta_j, \psi_i)$$

$$M^{\theta_i}(\Theta) = 1.0 - \text{ES\_compatibility}(\theta_j, \psi_i) - \text{ES\_incompatibility}(\theta_j, \psi_i)$$

However, if the total confidence is greater than 1.0, then the following SEF will be produced.

$$M^{\theta_i}(\theta_i) = \frac{\text{ES\_compatibility}(\theta_j, \psi_i)}{\text{Conf}_{tot}}$$

$$M^{\theta_i}(\neg\theta_i) = \frac{\text{ES\_incompatibility}(\theta_j, \psi_i)}{\text{Conf}_{tot}}$$

$$M^{\theta_i}(\Theta) = 0.0$$

Again, the masses for all other propositions will be zero.